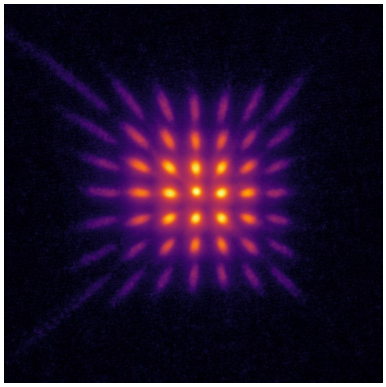
Characterisation of a
Cold-Atom Electron Source
for
Ultrafast Diffractive Imaging

Joshua Stephen Jones TORRANCE

ORCID: 0000-0002-0463-2967



Submitted in total fulfilment of the requirements of the degree of Doctor of Philosophy

School of Physics
The University of Melbourne

November 14, 2018

Supervised by Professor Robert E. Scholten and Doctor Benjamin M. Sparkes

Abstract

The understanding of atomic structures and processes is continually improving with the great technological development in imaging techniques. Ultrafast electron and X-ray techniques are able to perform measurements at atomic lengths and timescales and both these techniques require the generation of high-brightness ultrashort-duration electron bunches. Electron imaging techniques directly use these short bright bunches and in X-ray free electron lasers (XFELs) the electron bunches are used to generate short bright bunches of X-rays.

It is hoped that cold-atom electron and ion sources (CAEISs) will also be able to produce ultrashort high-brightness electron bunches that are brighter than conventional sources. CAEISs generate electrons via near-threshold ionisation from an ultracold atomic gas and have been shown to create electrons bunches with temperature as low as 10 K. Conventional photocathode sources have temperatures of thousands of Kelvin and, as brightness is inversely proportional to the temperature of the source, CAEISs have the potential to produce much brighter electron bunches. CAEISs are also capable of producing extremely cold ion bunches and show great promise as an ion source for ion milling and microscopy.

This thesis describes a number of developments involved with the CAEIS project at the University of Melbourne, in particular pushing the boundaries of laser frequency stabilisation to allow for precise selection of atomic states for cooling and ionisation, and a new technique for measuring the brightness of charged particle bunches.

Laser frequency stabilisation is an essential component of the CAEIS and many other applications including metrology, spectroscopy and laser cooling. Polarisation spectroscopy is a commonly used technique for laser frequency stabilisation but the full measurement and control bandwidth has not previously been demonstrated. Here it is shown that the bandwidth is sufficient to not only stabilise the frequency of the laser, but also to reduce the laser linewidth to much less than 1 kHz, two orders of magnitude better than previously reported. This demonstration provides a new approach for precisely accessing the high-lying Rydberg-levels of atoms, if used in conjunction with cavity based frequency locking methods, allowing for a greater exploration of the ionisation methods involved in a CAEIS.

Brightness is the most comprehensive figure of merit for charged particle beams and a new technique for measuring beam brightness with sub-nanosecond time-resolution is presented. The technique achieves time-resolved brightness measurements by streaking one-dimensional

pepperpot measurements across the detector. Time-resolved brightness measurements have the potential to reveal information related to the ionisation processes used in CAEISs and can show the efficacy of techniques used to counter the effects of space charge in the beams produced from a CAEIS.

The performance of the CAEIS apparatus operating in its normal pulsed mode is compared to continuous operation with emphasis on the beam current and electron trajectory stability. The beam quality was also improved by identifying an astigmatism in the beam and correcting it with a 3D printed magnetic quadrupole lens. Virtually all the beam measurements presented here utilise image processing techniques that allow for multi-shot averaging despite instability in the electron beam trajectory.

This iteration of a CAEIS was able to produce ultrashort-duration electron bunches and these have been used to demonstrate ultrafast electron diffraction from thin gold foils. This is an important step along the path to being able to perform ultrafast single-shot coherent diffractive imaging (CDI) and the next iteration of the CAEIS should have sufficient current to demonstrate diffraction that is both single-shot and ultrafast.

Declaration

I declare that:

i. The thesis comprises only my original work towards the PhD except where indicated in the preface.

ii. Due acknowledgement has been made in the text to all other material used.

iii. The thesis is fewer than 100,000 words in length, exclusive of tables, maps, bibliographies and appendices.

Joshua Stephen Jones Torrance

Contributions

This thesis presents work I was involved with as a member of the Atom Optics group in the School of Physics at the University of Melbourne. All work described in this thesis was conducted by me, unless mentioned below:

- The Cold-Atom Electron and Ion Source: The experimental apparatus was designed, constructed, and improved by many members of the group over a number of years. Some of the members of the group that have worked on the apparatus are Andrew McCulloch, Dene Murphy, Corey Putkunz, David Sheludko, Sebastian Saliba, Ben Sparkes, Rory Speirs, Richard Taylor, and Daniel Thompson.
- The results described in Chapter 4 built upon work done by Rory Speirs, published in Reference [1], and conducted in collaboration with Rory.

The research described in this thesis was supported by the Australian Research Council through Discovery Project DP170101148 and the Linkage Projects LP130100857 and LP150101188.

Contents

List of Figures

Chapter 1

Introduction

Molecular imaging has provided science with great advances, such as the determination of the helical structure of DNA in 1953 [2], and the discovery of the structure of myoglobin and haemoglobin in 1962 [3]. The cold-atom electron and ion source (CAEIS) at the University of Melbourne is an ongoing project that aims to develop a source capable of achieving the 'holy grail' of scientific imaging, the *molecular movie* [4, 5]: that is, a sequence of images with atomic spatial and temporal resolution. A CAEIS also has enormous potential for ion beam technologies such as focused ion beam milling machines (FIBs) [6], ion microscopes [7], and as a source for accelerators such as synchrotrons and X-ray free electron lasers (XFELs) [8–10].

The research described in this thesis has contributed to the development of the cold-atom electron source (CAES) towards future ultrafast electron diffraction (UED). This chapter provides the context to facilitate understanding of what is needed from a cold electron source and how the work in later chapters contributes to those requirements.

1.1 Making the 'Molecular Movie'

The ability to observe molecular dynamics at an atomic spatial and temporal scale could provide great insight into some of the basic processes of biology and chemistry. The fabled 'molecular movie' refers to capturing the atomic dynamics of molecular systems with atomic-scale spatial and temporal resolution as they undergo some transition such as a chemical reaction, protein folding, melting, or even just atomic vibrations. Molecular movies could provide greater understanding of important biological reactions such as photosynthesis or oxygen transport by haemoglobin.

One of the stepping stones towards molecular movies is single-shot imaging of non-crystalline molecules which would allow structural determination of membrane proteins, an essential component in rational drug design [11–13]. The majority of imaging performed to date has been on crystalline targets and the related imaging techniques are much better developed than those for

non-crystalline targets. One of the main techniques in structural determination of crystalline targets is electron diffraction which, along with X-ray and neutron methods [14, 15], has been utilised in structural determination of many materials with atomic resolution. Continuing research into real-space and diffraction imaging techniques is aiding in the understanding of a growing range of samples as well as providing new types of information on the samples under investigation.

In the past the greatest success with structural determination has been achieved with crystallography, limited to samples that can be crystallised. There are a large number of biological proteins of interest, particularly membrane proteins, that cannot be crystallised and developments in ultrafast imaging techniques are providing a route towards structural determination without crystallisation [16, 17]. Cryogenic electron microscopy is also making headway in this area but is unable to measure dynamics [18, 19]. Ultrafast imaging techniques are a relatively recent development and they provide the opportunity to study electronic and atomic dynamics. Ultrafast techniques allow for imaging to be completed before damage to the sample from the illumination makes imaging impossible [20–22]. Both X-ray and electron imaging techniques are limited by the capabilities of their sources which are undergoing continual development. The techniques using X-rays and electrons each have different advantages and disadvantages with regards to dynamic imaging and structural determination and they often give complementary information.

One proposed method for generating molecular movies involves using a high-brightness ultrashort duration beam source, such as an XFEL or future CAES, to perform coherent diffractive imaging (CDI) on individual molecules [4, 20, 23]. The individual molecules would be dropped one-by-one in front of the illuminating bunches which would be short enough that imaging is completed before damage from the illumination affects the diffraction pattern (see Figure 1.1). Dynamic processes, such as phase transitions, melting, and pumped vibrational modes, can be studied with this technique if the process can be triggered, say with a precisely timed laser pulse. By varying the delay between triggering the process and imaging, a 'movie' can be constructed. The random orientation of molecules as they are imaged can be managed algorithmically as long as there is sufficient brightness with each shot [24].

XFELs have been able to produce molecular movies [25–27] but there is still motivation for attempting to develop electron source alternatives. Electrons interact much more strongly with matter than X-rays, with interactions having cross-sections 10^4 to 10^6 larger [5], and thus much fewer electrons are required per bunch than X-ray photons. While an XFEL requires a several kilometre long facility with a billion dollar price tag a hypothetical electron source with similar capabilities would likely fit in a room with a much lower cost: current CAES implementations easily fit on a single optical bench.

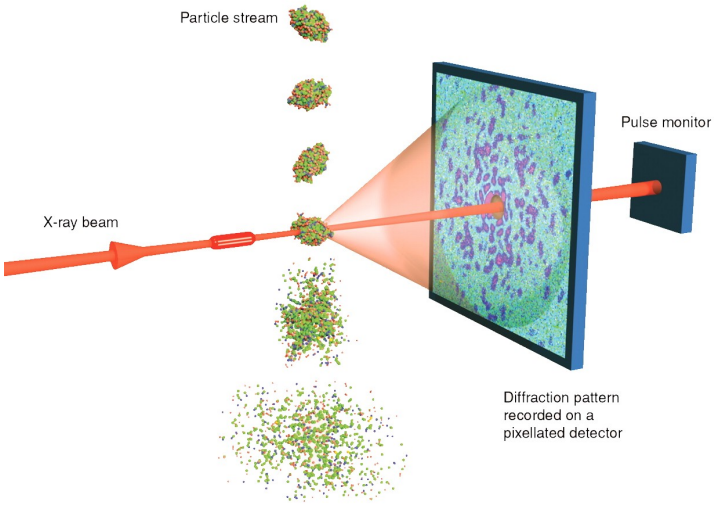


Figure 1.1: Structural determination of single molecules may be possible if a sufficiently bright and short pulse of X-rays is used to image the molecule before damage affects the diffraction pattern produced. Electrons could be used in place of X-rays if a suitable source can be developed. Image adapted from Reference [20].

1.1.1 Ultrafast Coherent Diffractive Imaging

Resolving the structure of single molecules requires sufficient signal such that the orientation problem can be solved and averaging performed, and the imaging needs to be completed before the molecule is substantially damaged by the beam [28]. Thus a single pulse must be extremely intense, in one scenario requiring 10^{12}, 8 keV X-ray photons or 10^{6}, 3 MeV electrons, and extremely short in duration, tens of femtoseconds [23, 29]. Damage to molecules can occur via numerous mechanisms and are different for X-rays and electrons but in both cases occur in approximately tens of femtosecond timescales [29].

The coherence of the source is also an important consideration for diffraction imaging as the beam must have a coherence length as large as that of the molecule under consideration so that portions of the illuminating wave diffracted from different positions on the molecule interfere coherently. When performing coherent diffractive imaging (CDI) the diffraction pattern detected is the square of the Fourier coefficients that represents the molecule under examination. Due to the loss of the complex phase components when the Fourier coefficients are squared it is not possible to directly invert to recover the structure of the molecules; this is known as the *phase problem* [30]. The most common solution to the phase problem is iterative computational phase-retrieval [31].

Ultrafast CDI has been demonstrated previously on micron-scale objects using an XFEL [23] and atomic scale targets with electrons [32]. Conventional electron sources have insufficient brightness for ultrafast CDI with electrons and hence there is a great interest in novel sources such as the CAEIS.

1.1.2 Imaging Targets

Crystallography has been of enormous benefit to biology as the structure of biological molecules plays a vital role in their function, with proteins interacting with sub-structures of other proteins to mediate biological processes, somewhat analogous to a lock and key, or jigsaw pieces. Knowing the structure of biological proteins is essential to fully understanding how the mechanisms that a protein is involved in functions and can provide vital information to researchers on how to produce drugs to manipulate those mechanisms [33, 34]. The design and creation of drugs based on knowledge of the structure of the proteins involved in a mechanism is sometimes referred to as *direct drug design* and has had some success to date [35–37]. The first example of structurally informed direct drug design was the drug dorzolamide which was released to the market in 1995 to reduce intraocular pressure in certain circumstances [38].

One of the obvious restrictions on crystallography is that the target sample must be crystallisable in order for it to be imaged and unfortunately there are large numbers of important biological proteins that scientists have been unable to crystallise, notably a large fraction of membrane proteins which mediate interactions on the surfaces of cells [39]. Alternative structure determination techniques such as ultrafast CDI with an XFEL or CAES would be able to bypass the crystallisation requirement and thus provide researchers with a wealth of useful information.

Knowledge of the changes in molecular structures during many complex and interesting dynamics (such as melting, photosynthesis, molecular phase transitions, or chemical reactions) would prove invaluable to understanding the physics, chemistry and biology of many areas.

1.2 Cold-atom electron and ion sources

The research described by this thesis is part of an ongoing effort to develop an alternative source of electrons and ions that extracts the charged particles from laser-cooled atoms. Initially the aim of the project was to create ultrafast coherent electron bunches for use in diffraction imaging in a similar way to ultrafast X-ray pulses, but it was later realised that a CAES could also serve as an injection source for particle accelerators or FIB devices. By simply reversing the polarity of the accelerator the source can generate high quality ion bunches with similar properties to the electron bunches thus providing a new source for use in ion microscopy and nano-fabrication.

Cold-atom sources operate by carefully ionising atoms in a magneto-optical trap (MOT) such that the resulting ions and electrons are cold and thus the bunches accelerated from those clouds have low emittance and high coherence. A brief schematic of CAEIS operation is shown in Figure 1.2. The low transverse temperature of ions and electrons produced by a CAEIS is one of the main advantages of this source. The source also allows for the production of ultrafast, and arbitrarily shaped electron and ion bunches. These techniques are applicable to any of the

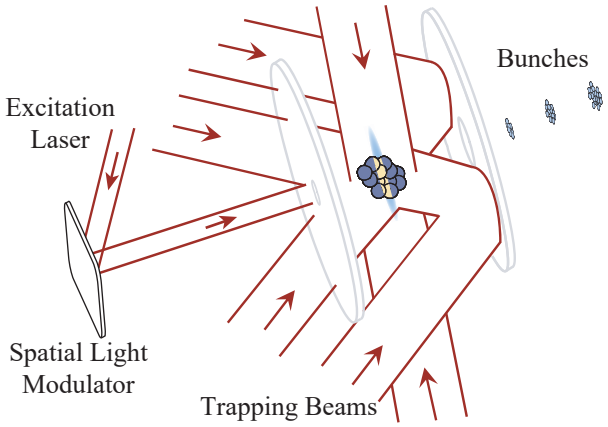


Figure 1.2: Rubidium atoms are trapped and cooled in a MOT before being ionised with red and blue laser beams. The electrons or ions are then accelerated by two electrodes, forming the electron or ion bunches. In this diagram the blue ionisation laser comes from behind the atom cloud.

many atomic species that can be laser cooled and trapped.

A CAES can be thought of as a photocathode electron source with the normally solid cathode replaced with an ultracold gas which provides the CAES with a few advantages such as high quantum efficiency, and near-threshold ionisation producing colder electrons than those from other photocathode sources [40]. The simplicity of the interactions between photons and atoms allows for high quantum efficiency [41]. Another advantage of gas photocathodes is the lack of optically induced damage from high-intensity laser-fields. Solid cathodes undergo constant degradation and require regular replacement [42] whereas the gaseous target in a CAES is renewed with every bunch produced.

A major advantage of a CAEIS as an electron or ion source is the low temperature of the charged bunches, made possible by precise ionisation of atoms trapped in a MOT. The ultracold atoms trapped in the MOT have temperatures around 100 μK but after ionisation the electrons have temperatures determined by the excess energy from the ionisation process [43–46]. Precise control over the photoionisation is possible by wavelength tuning the narrow-band ionisation lasers, allowing the excess energy from the ionisation process to be minimised. The electrons produced by the source can have transverse temperature as low as ~10 K [47]. Ions produced by the source have their initial temperature determined predominantly by the temperature of the trapped atoms before ionisation with an ion temperature around 1 mK [48, 49].

The ionisation methods used in a CAEIS involve a two-step ionisation process that first excites the atoms to an intermediate state with one laser followed by near-threshold ionisation of the excited atoms with a second laser frequency-tuned to minimise the excess ionisation energy. Generally the second stage in the excitation process involves exciting the atom from

the intermediary excited state to a high-lying Rydberg state from which the atom can be field ionised in the presence of the accelerating electric field [50]. Due to the small energy difference between high-lying Rydberg states narrow linewidth lasers are required. To create lasers with sufficient precision, polarisation spectroscopy frequency feedback control has been explored (Chapter 3) [51, 52]. Polarisation spectroscopy with high-bandwidth feedback is able to provide laser frequency linewidths two orders of magnitude smaller than previously reported. Polarisation spectroscopy will prove useful for precision interaction with the high-lying Rydberg levels used in CAEIS ionisation, when combined with techniques such as Pound-Drever-Hall (PDH) frequency stabilisation, as well as in other fields such as spectroscopy or metrology where low laser linewidth is important.

The sophisticated ionisation scheme used in a CAEIS allows for the production of ultrashort duration bunches and arbitrary shaping of the bunches. The CAEIS has been shown to produce electron bunches with duration less than 130 ps [46] and should be able to provide bunches suitable for compression to bunch duration of order 100 fs, suitable for ultrafast imaging [53].

The CAEIS also has the ability to produce arbitrarily shaped electron or ion bunches by manipulating the profiles of the laser beams used to ionise the atoms [54]. This capability is useful in a large number of ways but has particular potential as an avenue for the production of uniform ellipsoidal bunches to allow for reversal of beam-quality degradation by space-charge expansion [55, 56].

One of the most important figures of merit for charged particle beams is beam emittance which can be directly correlated with the temperature of the particle source. Emittance describes the angular spread of particles within a beam or bunch and can be thought of as the "focusability of the beam" with low emittance being preferable. Due to the extremely low temperature, the CAEIS has enormous potential as a source for low-emittance electron and ion bunches. Low emittance is also related to high coherence which is a vital consideration for imaging. The coherence length of a source must be at least as large as that of the sample under consideration for techniques such as CDI or at least as large as the unit cell length for crystallographic diffraction techniques. The CAEIS has already demonstrated impressive coherence lengths as large as those of some small biomolecules with further improvements expected [47].

A new technique for measuring beam brightness (a metric that combines beam emittance and current) with time resolution was developed. The new method operates by streaking one-dimensional pepperpot measurements across the detector [57], see Chapter 5. This approach was developed to allow for examination of the efficacy of space-charge reversal techniques when they are implemented with a CAEIS and to learn more about the ionisation processes involved in a CAEIS. This technique is generally applicable to charged particle beams, not just the CAEIS, and could prove useful in numerous applications.

The CAES has been used to demonstrate single-shot diffraction [1] and ultrafast diffraction (see Chapter 4) from a gold nanofilm. To date diffraction that is both single-shot and ultrafast

has not been achieved due to the low beam current of this CAES implementation but there are a number of strategies for improvement, some of which are explored in Chapter 2. If the beam current of the CAES can be improved then there is enormous potential for the source as it will produce high-brightness, ultrashort-duration electron bunches with a direct method to counter the effects of space-charge expansion. The CAES shows great promise as a source for CDI and the production of molecular movies.

1.2.1 Ion Source

By reversing the polarity of the acceleration stage of beam production in a CAEIS a beam of ions can be produced instead of an electron beam. Ion beams share the same advantages as the electron beams with bunches being shapeable, coherent and low emittance [7].

Focused ion beams (FIBs) have a wide variety of applications such as high-resolution imaging [58], sample analysis and nanofabrication [59]. Liquid metal ion sources are the most commonly used FIB sources, usually using gallium ions for simplicity and robustness despite gallium having a tendency to contaminate or destroy samples. A cold-atom ion source (CAIS) promises to provide an attractive alternative to conventional sources as it would have high brightness and low emittance, and would be able to operate with a large range of ions, which could be selected to avoid sample contamination.

CAISs have been used to demonstrate microscopy with lithium ions [60] and chromium ions [61]. Rubidium CAIS ion beams have been characterised and used to demonstrate the suppression of space-charge expansion [49, 62]. A caesium CAIS has been use to demonstrate resolution better than commercial sources, with measured brightness more than 24 times greater than the highest brightness observed in a gallium liquid metal ion source [63].

1.3 Summary

Molecular structural determination of biological proteins is an essential component of modern developments in medicine and new tools are constantly required to extend the boundaries of the field. The CAES has great potential as an alternative source for molecular imaging and may one day challenge existing sources such as XFELs, photocathode electron sources, and cryoelectron microscopes. Current iterations of the CAES are able to produce high-brightness, ultrashort-duration bunches with coherence lengths equal to that of small biomolecules and if the beam current can be improved in the next generation of CAES then they will be well on the way to becoming a competitive source for the production of molecular movies.

The CAEIS operating in ion mode also shows much promise in FIB applications. CAIS technology may well be applicable to any atom that can be optically trapped which will provide significantly more scope to the application of FIBs.

This thesis discusses a number of pieces of research conducted with the iteration of the

CAEIS located at the University of Melbourne that operated until 2018. Chapter 2 explains the CAEIS apparatus and details some characterisation of and improvements to the apparatus with the aim of improving the quality of beams produced by the source. Research that improved the performance of polarisation spectroscopy as a frequency stabilisation technique, demonstrating sub-kilohertz linewidth, is detailed in Chapter 3 (published as Reference [52]) and should prove useful in the ionisation portion of CAEIS operation. Chapter 4 presents diffraction results from the CAES which are the first demonstration of ultrafast diffraction from gold foil with a CAES, reaching an important milestone on the route towards UED and the molecular movie. A new method for measuring time-resolved emittance by streaking one-dimensional pepperpots is presented in Chapter 5 and published in Reference [57] providing a useful technique for investigating the ionisation processes involved in a CAEIS and characterising the performance of techniques to counter space-charge degradation of bunches produced by a CAEIS.

Chapter 2

Cold-Atom Electron and Ion Source

The CAEIS at the University of Melbourne is a source of low temperature electrons or rubidium ions with promising potential as an alternative charged particle source. The CAEIS works by carefully ionising rubidium atoms trapped in a MOT to generate a low temperature plasma which can then be electrostatically accelerated to form a bunched particle beam. The apparatus described here has enabled numerous experiments which have provided greater understanding of certain aspects of atomic physics and the capabilities of the CAEIS. This work identifies some of the strengths and weaknesses of our CAEIS and provides guidance for developing the next generation of cold atom sources.

The CAEIS was initially designed and developed by then doctoral students Simon Bell and David Sheludko and is discussed in detail in References [64] and [65]. Numerous other masters and doctoral students have worked on maintaining and improving the system and details can be found in their theses [66–71].

The description of the CAEIS in this chapter provides a context for the rest of the thesis as all research conducted was directly or indirectly related to the continued improved of the CAEIS. Section 2.1 provides a description of the apparatus and some of its limitations. A number of investigations and improvements to source stability, source current and beam optimisation are also presented in this chapter. A comparison of the standard pulsed operation is compared to continuous operation in Section 2.2 to explore the possibility of achieving greater time-averaged beam current. An essential component of data processing with this apparatus is the technique used for managing multi-shot image averaging with the unstable electron trajectories, called 'registration', and this is discussed in Section 2.3. Without registration the signal-to-noise ratio (SNR) in most of the results described in this thesis would be worse and a number of the measurements performed would be impossible. Section 2.4 describes the characterisation and correction of an astigmatism in the electron bunches which provides improved beam quality and SNR for the results described in later chapters.

2.1 The Melbourne CAEIS

The first step in generating electrons and ions at the University of Melbourne CAEIS was trapping and cooling atoms in a MOT loaded from a Zeeman slower. The optical and magnetic trapping fields were then extinguished and the cold ground-state atoms ionised using a combination of a red excitation laser and a blue ionisation laser. The charged particles were accelerated by a static electric field produced by the accelerator electrodes, one polarity accelerating ions towards the detector and the other electrons. The main detector for experiments was a phosphor-coupled microchannel plate (MCP) combined with a charge-coupled device (CCD) camera.

When acting as an electron source it was essential to turn off the MOT and Zeeman slower magnetic fields before ionisation due to the significant deviation that the magnetic fields cause to the electron trajectories. Ion trajectories are not adversely affected if the fields are left on due to the much higher mass of ions.

There were a number of light fields required for the running of the CAEIS. The 780 nm red lasers[1] were frequency locked relative to the rubidium-85 primary cycling transition, $5^2S_{1/2}$ $(F = 3) \rightarrow 5^2P_{3/2}$ $(F' = 4)$, also known as the cooling transition, and the repump transition, $5^2S_{1/2}$ $(F = 2) \rightarrow 5^2P_{3/2}$ $(F' = 3)$. The repump beams listed below were mixed with the cooling beams and coupled into the same optical fibres for delivery to the vacuum system. The following beams were used:

- *Zeeman Slower Beam:* Used in conjunction with the tapered magnetic coil to slow atoms leaving the oven. This beam was locked 250 MHz below the cycling transition.
- *Zeeman Slower Repump Beam:* Used to keep atoms in the Zeeman slower from being lost to the dark $F = 2$ ground state. Locked 250 MHz below the repump transition.
- *MOT Cooling Beams:* Used to cool and trap lasers in conjunction with the MOT magnetic field. Locked 10 MHz below the cooling transition.
- *MOT Cooling Repump Beams:* Used to keep atoms in the MOT from the dark state where they can no longer be trapped. Locked to the repump transition.
- *Continuous wave (CW) Excitation Beam:* Used to excite atoms in the MOT in the first stage of ionisation. Locked to the cooling transition.
- *CW Excitation Repump Beam:* Used to keep atoms out of the dark state when performing ionisation. Locked to the repump transition.
- *Femtosecond Excitation Beam:* Generated by a mode-locked Ti:sapphire amplified pulse laser[2] with 780-830 nm wavelength and a minimum pulse duration of 35 fs.

[1] MOGLabs External Cavity Diode Lasers with MOGLabs Diode Laser Controllers

[2] Spectra-Physics Spitfire Pro

- *Pulsed Ionisation Beam:* A tunable 457-493 nm 10 ns-pulse laser used to perform the second stage of ionisation[3].

- *CW Ionisation Beam:* A CW tunable 480 nm laser used as an alternative to the blue pulse laser[4].

- *Imaging Beam:* Used to image the atom cloud. Locked 4 MHz below the cooling transition.

A previous iteration of the cooling and repump laser systems for the CAEIS consisted of six separate diode lasers each independently locked to appropriate rubidium transitions. That setup suffered in reliability as, if any of the six diodes became unlocked (a common occurrence) then identification and relocking of the offending laser was a lengthy process. The setup was streamlined by using only two diode lasers amplified by tapered amplifiers (TAs) with greater resistance to environmental perturbations making the loss of lock a less common occurrence and easier to recover from. A simplified schematic of the cooling and repump laser setup is shown in Figure 2.1.

2.1.1 Rubidium Oven

The source begins with an effusive rubidium oven with a long heated collimation tube. Typically effusive ovens are wasteful with large numbers of atoms lost to large solid angles. This apparatus makes use of a long heated collimation tube to collect and re-emit atoms that were initially emitted at high angles. These atoms are re-emitted back to the reservoir or into the collimated atom beam leaving the collimation tube. The rubidium reservoir was typically heated to 80 °C and the collimation tube to 120 °C. Detail on the design, operation, and performance of the oven can be found in References [72] and [65].

2.1.2 Zeeman Slower

A Zeeman slower using a solenoid with tapered winding pitch is used to slow the atoms so that they can be captured by the MOT [72]. Zeeman slowers operate by using a laser red-detuned from resonance to slow the atoms down but as the atoms slow the conditions for resonance change due to the changing Doppler shift thus preventing further slowing of the atoms. The solution to this quandary used here was a tapered magnetic coil that applied a magnetic field to shift the atomic resonance such that a particular velocity class of atoms remained resonant with the light field, and thus was slowed, along the length of the Zeeman slower. Atoms leaving the Zeeman slower typically had velocities around 35 m/s, well within the capture velocity of

[3] Sirah dye laser system CSTR-D-3000

[4] Toptica TA-SHG pro

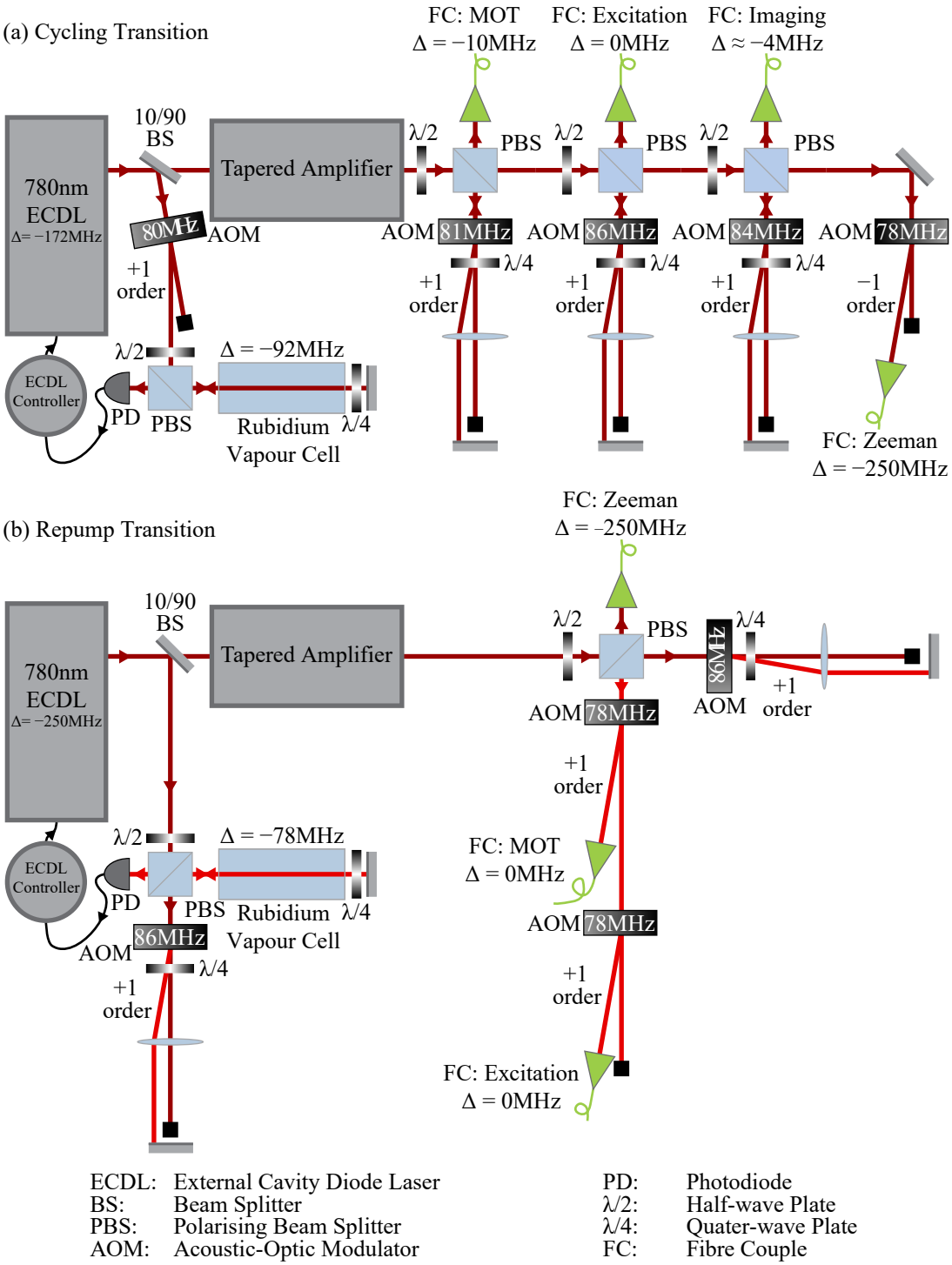


Figure 2.1: The simplified setup of the lasers locked to the rubidium-85 cycling (a) and repump (b) transitions. The external cavity diode lasers (ECDLs) are locked with saturated absorption spectroscopy, amplified with tapered amplifiers and frequency-offset using acousto-optic modulators (AOMs). Δs refer to the frequency relative to the cycling or repump transitions.

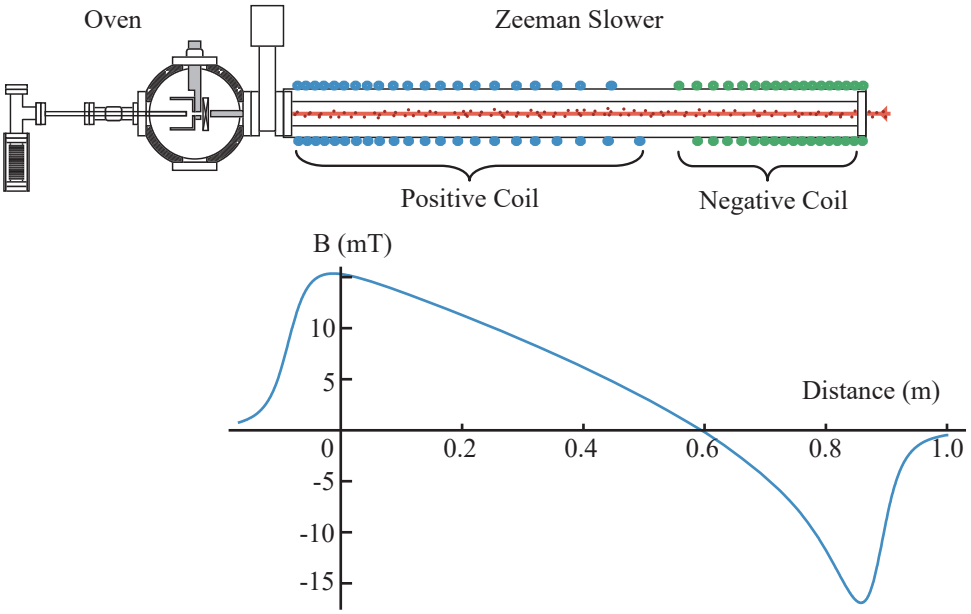


Figure 2.2: A schematic of the rubidium atom source. Atomic vapour from the oven is directed into the Zeeman slower where a laser detuned from the atomic resonance (shown in red), in combination with a tapered magnetic coil (blue and green), with a magnetic field as shown, slows the thermal atoms.

the MOT. A schematic of the Zeeman slower along with the magnetic field produced by the tapered coil is shown in Figure 2.2.

When extracting electrons from the MOT the magnetic trapping coils must be turned off to prevent disruptions to the electron trajectory.

2.1.3 Magneto-Optic Trap

Magneto-optical traps (MOTs) use a combination of magnetic and light fields to trap and cool atoms to μK temperatures [73]. In the CAEIS the MOT was formed from six counter propagating 780 nm lasers in a retro-reflective quasi-mirror MOT configuration [67, 74] to allow for the accelerator structure which consisted of transparent and reflective electrodes, as shown in Figure 2.3. The magnetic field component of the MOT was formed from two magnetic coils in an anti-Helmholtz configuration providing a zero-field region in the centre of the trap.

The MOT trapping lasers were detuned -10 MHz from the cycling transition and were mixed with on-resonant light at the repump transition to pump atoms that fell into the dark $F = 2$ state.

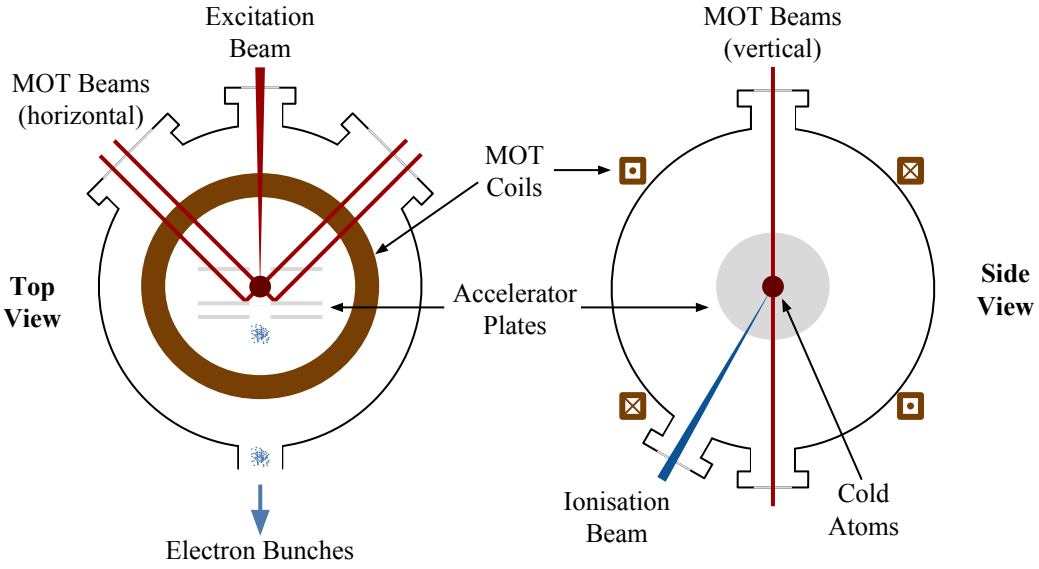


Figure 2.3: A diagram of the magneto-optic trap, ionisation lasers and accelerator structures.

2.1.4 Ionisation

The CAEIS was capable of generating bunches of electron and ions with long (~10 µs), short (~5 ns), and ultrashort (~10 ps) bunch duration depending on the laser systems used [46, 71]. Depending on the ionisation pathway, the CAEIS was able to produce cold electron bunches (~10 K) or hotter electron bunches (>10 K). Generally cold electron bunches were preferable and desired bunch duration depended on the application, UED for example requires ultrashort bunches.

Rubidium has a ground state ionisation threshold of 4.18 eV which can be generated using one blue and one red photon. Red light could be generated by a CW diode laser amplified by a TA and locked to the cycling transition or it could be generated with a mode-locked Ti:sapphire amplified pulsed laser with a wavelength range of 770 nm to 830 nm and a minimum pulse length of 35 fs. Blue light could be generated with a tunable dye pulse laser that produced 460 to 490 nm light with a full-width half maximum (FWHM) duration of 5 nm or with a CW laser generated by a high-power tunable frequency-doubled diode laser.

Sequential ionisation utilised a single red photon to excite from the ground state to an intermediate excited state followed by a single blue photon to transition to a field-ionising state or to the ionisation continuum. For sequential ionisation, the bunch duration was determined by the shortest of the duration of the laser pulse driving the transition from the exited state to the ionising state, the lifetime of the intermediate state, or by the depletion time of the intermediate state.

Multiphoton excitation occurs when the laser intensities are high enough for nonlinear optical transitions to occur which was the case when the red or blue pulse lasers were tightly focused into the atom cloud. In multiphoton excitation two or more photons are absorbed

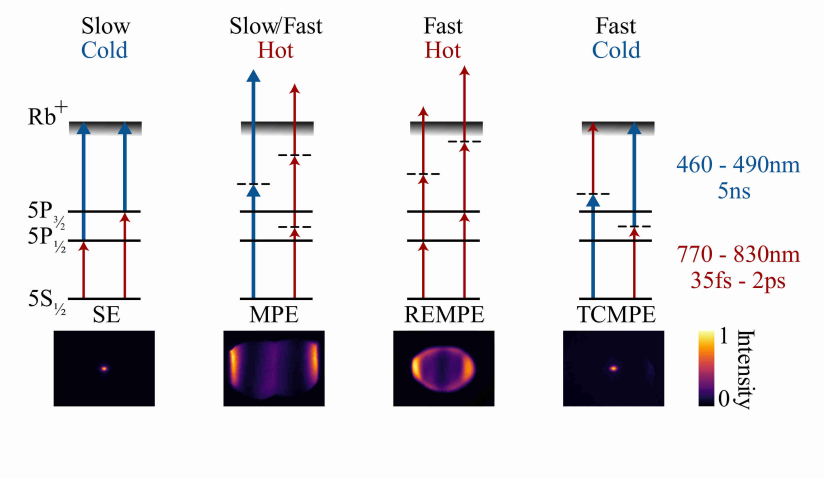


Figure 2.4: A number of photoexcitation pathways were possible in the presence of high intensity illumination by the red a blue lasers in the CAES such as sequential excitation (SE), multiphoton excitation (MPE), resonance-enhanced multiphoton excitation (REMPE), and two-colour multiphoton excitation (TCMPE). TCMPE is the only pathway that produces bunches that are cold and ultrashort. The images show the relative transverse momentum distributions for the detected bunches. This figure is from Reference [71].

without transitioning via a real intermediate state. If n is the number of photons absorbed for the atom to reach its final ionised state then the transition rate is proportional to the nth power of the optical intensity [75]. Due to the short lifetime of the virtual intermediate states the bunch duration is determined by the duration of the laser pulses. Multiphoton excitation can occur with just photons of one colour or with photons of both colours.

Two-colour multiphoton excitation (TCMPE) occurs when one blue and one red photon are absorbed but the intermediate state is a virtual state. Due to the short lifetime of the intermediate state the duration of bunches produced by TCMPE depends on duration of the shortest laser pulse.

Resonace-enhanced multiphoton excitation (REMPE) occurs when there is a combination of sequential excitation and multiphoton excitation. Here a number of photons are absorbed to excite the atom to a real intermediate state followed by more photons of the same colour being absorbed to transition to the final state. As less photons are required for each transition the overall transition rate can be much higher when compared to plain multiphoton excitation.

The temperature of particles generated from the CAES depends primarily on the excess ionisation energy given to the atoms by the absorbed photons. Due to the complex spatial probability distributions of electrons in high-lying states the relationship between absorbed photon energy and temperature is complex [76] but it is generally true that with greater photon energy comes greater source temperature. The classical ionisation threshold is lower in the presence of electric field, such as the accelerating field in the CAEIS, due to the Stark-shift.

The excess energy of an ion or electron relative to the classical ionisation threshold is:

$$\Delta E = -E_I + 2\sqrt{ke^3F} + \sum_{i=0}^{n} \frac{hc}{\lambda_i}, \tag{2.1}$$

where $E_I = 4.18\,\text{eV}$ is the ground state ionisation energy of rubidium-85, k is the Coulomb constant, e is the elementary charge, F is the strength of the electric field, h is the Planck constant, and c is the speed of light. The second term represents the Stark-shift of the classical ionisation threshold, and the last term is the sum of the energy of the n photons involved in the ionisation, with wavelengths λ_i. Equation 2.1 assumes that rubidium is hydrogen-like which is a good approximation as long as $E_I \gg 2\sqrt{ke^3F}$ [77].

Sequential excitation and TCMPE are the only ionisation processes that produced cold electrons as the atoms underwent field ionisation due to the combined photon energy being slightly less than E_I. With the other methods the combined photon energy is larger than E_I and the excess ionisation energy is transferred to the electrons. The electron bunch temperature was demonstrated experimentally [46] and can be seen in Figure 2.4 in the momentum distributions for various ionisation processes.

The duration of the charge particle bunches depended on a number of factors [46]. Direct ionisation, with combined photon energy greater than E_I, resulted in short duration bunches with duration equal to that of the blue laser driving the transition to the ionisation continuum, 5 ns. Excitation to a Rydberg state, followed by field ionisation, resulted in relatively long duration bunches as the bunch duration in this case depends on the lifetime of the Rydberg state, typically 10s of microseconds. Ionisation by TCMPE could result in ultrashort duration bunches, 10 ps, if the red femtosecond-duration pulse laser was used to excite the atoms to the virtual intermediate state.

Beam Shaping

The CAEIS was able to shape the profile of the electron and ion bunches produced by manipulating the red and blue ionisation laser beams [54]. The lasers imparted their spatial profiles onto the ionised atoms which, for cold low-emittance bunches, could be maintained to the detector. The transverse bunch profile was determined by the red excitation laser profile combined with the density profile of the atomic cloud. Similarly the longitudinal profile depended on the ionisation laser profile and atomic cloud density.

Control over the excitation beam profile was achieved with an spatial light modulator (SLM) and an iterative feedback system [78]. With a second SLM and control system the longitudinal profile could also be controllable which would allow for full three-dimensional control over the beam shape. A schematic of the bunch profile system is shown in Figure 2.5 and an example of its performance can be seen in Figure 2.6.

Control over the bunch profile allows for customisation and feedback for whatever application the CAEIS is being used for but is of particular interest for countering space-charge

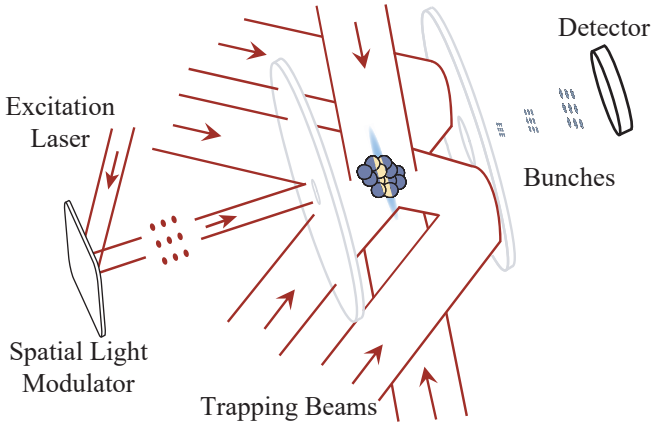


Figure 2.5: A schematic of the apparatus used to produce arbitrarily shaped bunches. The excitation laser pulse is shaped with an SLM to form an arbitrary beam profile at the atom cloud, imparting that profile onto the bunch produced by the CAEIS. In this diagram the blue ionisation laser comes from behind the atom cloud.

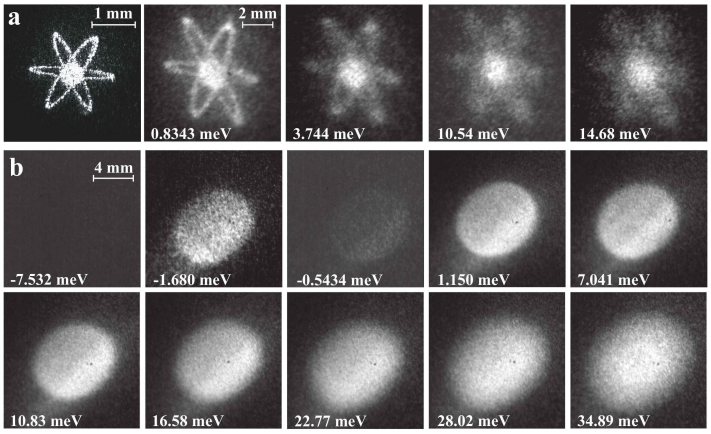


Figure 2.6: An example of beam shaping and the effects of temperature on the bunch profile. **a** The left most image indicates the excitation laser profile, and the other images are the transverse electron bunch profiles as excess ionisation energy is varied. **b** Transverse electron bunch profiles generated by a uniform excitation laser profile as excess ionisation energy is varied. The white text indicates the image dimensions and the excess ionisation energy in meV, with negative values being below threshold ionisation. At -1.680 meV excess ionisation energy the bunch current increases as the ionisation laser couples to a Rydberg state. This figure was adapted from Reference [54].

expansion. Self-interactions between charged particles within a sufficiently dense bunch, referred to as space-charge expansion or Coulomb expansion, results in degradation in the quality of a bunch. The loss of beam quality is usually expressed as an increase in emittance (see Chapter 5 for a definition of emittance). Increases in bunch emittance also results in a decrease in coherence which has ramifications for imaging applications. With the electron bunch charges achieved to date only the ultrashort duration electron bunches are dense enough to experience significant space-charge expansion. Ion bunches produced by this source are usually dense enough to demonstrate space-charge. Suppression of space-charge expansion has been demonstrated with the CAEIS by using beam shaping to produce bunches with linear and reversible Coulomb expansion [55, 62].

2.1.5 Accelerator

The static accelerating electric field was generated by two electrodes which were approximately 11 cm in diameter and 4 mm thick. One electrode was transmissive and anti-reflection (AR) coated at 780 nm and coated with indium-tin-oxide which is also transmissive to 780 nm to 96%. The second electrode was reflective and composed of polished gold-coated copper. Each of the electrodes had an aperture in the centre to allow the electron and ion bunches to pass through.

The limit to the beam energy the apparatus was able to produce, approximately 12 keV, was determined by the maximum voltages that could be applied to the electrodes; any higher and the electrodes would begin to either arc or short.

2.1.6 Beam Optics

There were a number of systems in place for manipulating the electron and ion beams: a solenoid lens, an Einzel lens, a magnetic quadrupole lens, a one-axis deflector, and a number of permanent magnets located outside the vacuum system. A schematic of the beamline is shown in Figure 2.7. The lenses were used to focus the bunches, usually to a focus on the detector but sometimes to manipulate the size of the beam at the sample plane. The quadrupole lens was used to counter astigmatism in the beam and is discussed in more detail in Section 2.4. The deflector was located after the sample stage and was used to streak bunches across the detector for the results presented in Chapter 5. The permanent magnets were used to steer electron bunches through the system, through a number of apertures and countering the deflection imposed by a number of anomalous magnetic fields present with the CAEIS.

2.1.7 Sample Management

Experimental samples were mounted on a custom sample mount formed from an aluminium paddle that was large enough to mount eight transmission electron miscroscopy (TEM) samples

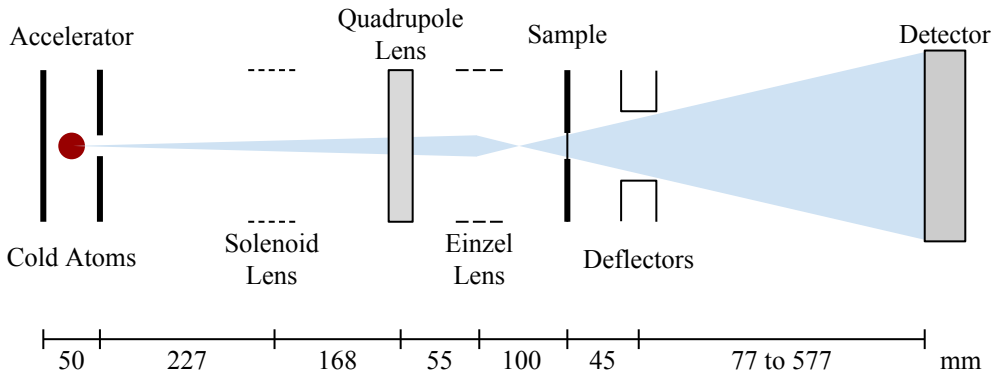


Figure 2.7: A schematic of the optics along the charged particle beamline in the CAEIS. The atoms are trapped in a MOT (not shown) located between two accelerator electrodes. Downstream from the accelerator is a magnetic solenoid lens, a magnetic quadrupole lens, an electrostatic Einzel lens, the sample holder, one-dimensional deflectors, and the MCP detector mounted on translating bellows with a z range of 500 mm. The numbers at the bottom of the figure indicate the distance between elements and the horizontal axis is not to scale.

and block any portion of the beam not passing through the samples. The samples were held by two commercial TEM mounts attached to the paddle that were able to fit 3.05 mm diameter samples with approximately 2 mm diameter of the sample visible to the bunches when the sample lid was in place. Grazing incidence reflection diffraction samples could also be mounted at the end of the paddle as shown in Figure 2.8. The sample paddle was mounted on a stage with manual control over two-axis translation transverse to the beam axis, as well as rotation about the horizontal transverse axis. The sample paddle could also be connected to a high-voltage supply to allow for the sample to be biased to further manipulate the beam energy as described in Section 4.2.1.

2.1.8 Timing

Synchronisation throughout the experiment was achieved with a 24-channel digital timing card[5]. Low voltage TTL signals were used to trigger the numerous time-sensitive devices involved in the experiment. The CAEIS operated at 10 Hz, the frequency of the flashlamp of the 5 ns blue pulse laser. The sequence began with MOT and Zeeman lasers and fields turning on to load the trap for approximately 90 ms. The trapping lasers and fields were extinguished 5 ms before the excitation and ionisation lasers and CCD camera were triggered to generate and detect the electron or ion bunch.

[5]Spincore Pulseblaster PCI mounted in external USB enclosure.

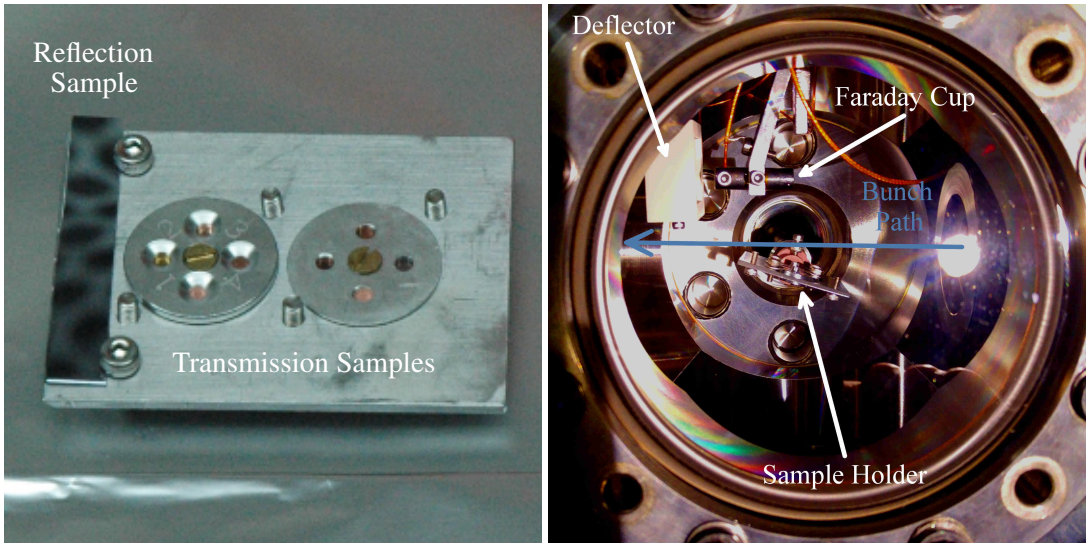


Figure 2.8: The sample holder used in the CAEIS. In the left photograph the sample holder with eight transmission samples and a reflection sample is shown. On the right is the sample holder in the sample chamber, with the post-sample deflector and Faraday cup retracted from the beam path. Bunches travel from right to left as shown by the blue arrow.

2.1.9 Current Limitations of the Melbourne CAEIS

There were a number of factors preventing the CAEIS achieving its goal of electron diffractive imaging that was both single-shot and ultrafast, most importantly the low electron beam current and the instability of the electron trajectory. These factors also require improvement for other CAEIS goals such as FIB milling.

The University of Melbourne CAES was able to generate bunches of 5×10^5 electrons when generating 5 ns duration bunches, which is of order the charge required for UED [53] but only a few hundred electrons when generating 10 ps duration bunches [46]. There are a few avenues available to the CAEIS to improve the bunch count such as increasing the MOT density, increasing the power of the ionisation laser, better optimisation of the ionisation pathways [50], and potentially running the apparatus in CW mode. CW mode has the potential to increase the number of particles per second however some applications would be difficult or impossible without discrete bunches. An investigation of the performance of the apparatus in CW mode is described in Section 2.2.

The stability of the source was also problematic, particularly for electron bunches, as a number of transient effects interfered with the path of the beams on timescales from shot-to-shot to minute-to-minute. Effects such as eddy currents, due to the switching of the strong magnetic fields, in the steel of the vacuum system and optical tables was one of the primary suspects for the perturbations in the beam path. Other potential sources are the electronic equipment such as power supplies and switches that were placed as far away from the beam

path as practical but were still in the vicinity. The poor stability of the source made some measurements difficult and others impossible. Multi-shot measurements were required in a number of cases and, due to the drift of the beam, straight averages of the images from the detector resulted in an unusable blur. Generating usable images for multi-shot measurements required 'registration' of the images, set averages where the features in each individual image was aligned with the other images to prevent blurring. Measurements without sharp features could not be well registered, as the registration algorithms have trouble precisely aligning the images. Registration is discussed further in Section 2.3.

2.2 Pulsed vs Continuous

This CAEIS had not previously been operated as a continuous source and with minor modifications it was adapted to act as a continuous source. Comparing the performance of the source in its usual pulse mode against the performance as a continuous source had the potential to show pathways for improving the pulsed mode. Continuous operation, if found advantageous, could have provided an alternative that could be used to demonstrate particular applications, such as coherent diffractive imaging (CDI), that do not require the source to be pulsed. Continuous mode functions by using the two-stage ionisation and acceleration on the atom beam from the oven without cooling or trapping by the Zeeman slower and MOT. It was hoped that the average beam current would increase with this mode of operation and that the beam produced would be subject to less drift and instability as the large magnetic fields were not switching. This investigation of continuous operation also, almost serendipitously, permitted a more detailed investigation of the instabilities in the electron trajectory as it provided an opportunity to examine the temporal behaviour of the instability, something not possible when the source operates at a mere 10 Hz.

While the entire apparatus and control system had been designed to operate in pulse mode it was fortunately relatively simple to modify the setup to operate to continuously produce an electron beam. The lasers and magnetic field of the MOT were disabled, the excitation beam was left on, and the CW blue laser was used instead of the usual blue pulse laser to drive ionisation.

Before switching to continuous mode the CAEIS had been measured to have 0.42×10^6 electrons per bunch equating to a beam current of 4.2×10^6 electrons per second. Initially the continuous mode was producing 0.64×10^6 electrons per second however there were a number of optimisations available to improve the current, as detailed below.

2.2.1 Oven Temperature to Electron Count

When operating in pulsed mode the MOT was given more than enough time to saturate and thus there was no benefit to producing more atoms from the oven. In continuous mode the

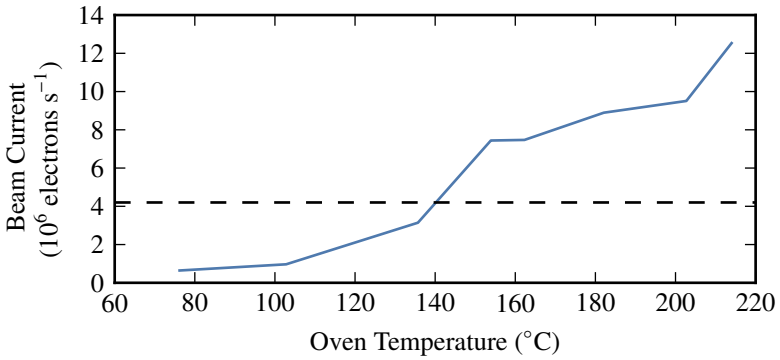


Figure 2.9: The electron beam current, measured by the calibrated MCP detector, compared to the temperature of the rubidium oven, as measured by an external sensor. The dashed black line indicates the approximate beam current for electrons generated with the CAES in pulsed mode.

beam current was dependent on the density of atoms in the ionisation region which could be increased by increasing the atomic flux from the rubidium oven.

The atomic flux from the oven could be increased by running the oven at a higher temperature. During normal operation the oven was set to a temperature of approximately 80 °C (rubidium has a melting point of 39.30 °C) however the oven was able to operate at up to 200 °C. Previous measurements indicate that the atomic flux from the oven was $5 \times 10^9\,\mathrm{cm^{-2}s^{-1}}$ at 80 °C and increased approximately linearly to $2 \times 10^{10}\,\mathrm{cm^{-2}s^{-1}}$ at 200 °C [72].

The effects of the oven temperature on the beam current are shown in Figure 2.9. There was a significant increase in the electron beam current as the oven temperature was increased. With the oven at a high temperature the beam current was greater than that of the pulsed mode bunches. An unfortunate side effect of running the oven at high temperatures is that the lifetime of the 5 g rubidium ampule in the oven will be reduced. Given the atomic flux measurements in Reference [72], running the rubidium oven at 200 °C will reduce the ampule lifetime by approximately a factor of four. This data indicates that the CW beam current could be improved with the use of transverse compression of the atom beam which would provide a similar improvement to beam current as more atoms would be present in the ionisation region [69].

2.2.2 Blue Laser Power

The second avenue available to increase the beam current is to increase the intensity of the CW blue ionisation laser beam. Optimisation of the laser and beamline allowed for up to 280 mW to be delivered to the vacuum system. By reducing the power of the blue beam incident on the atom beam and recording the beam current it was clear that the atoms were not saturated by the blue laser. If the blue laser power was saturating or close to saturating the atom beam then the

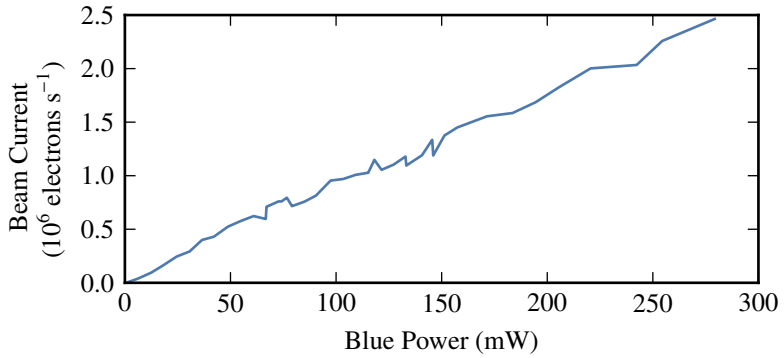


Figure 2.10: The electron beam current compared to the power in the CW blue ionisation laser with a rubidium oven temperature of 78.1 °C. The continuing upward trend indicates that the atom beam is far from saturated by the blue beam.

beam current would have been approaching a limit which is not apparent from the shape of the curve in Figure 2.10.

While the true situation was more complicated, involving the atom beam cross-sectional area and the blue laser intensity at the atom beam, the data shown in Figure 2.10 indicates that a more powerful blue laser or better optimisation of the laser intensity at the atom beam would increase the available beam current. Increasing the intensity of the blue ionisation laser would also improve the beam current when operating in pulsed mode as the available power of the blue pulse laser is poor. As well as optimising or replacing the laser systems, another avenue for increasing the intensity in the ionisation region would be to recycle the beam as most of the power is transmitted through the atom beam or MOT.

Using the maximum laser power available with an oven temperature of 200 °C provides approximately a sixfold increase in beam current compared to pulse mode.

2.2.3 Stability

The instability in the electron beam path observed in pulsed mode was still obvious with CW mode, indicating that the primary source of the instability was not the fast switching of the magnetic coils as had been previously suspected. The next most likely culprit for noise was something involving the mains power supply which operates at 50 Hz and thus, according to the Nyquist-Shannon sampling theorem, requires a sampling rate of at least 100 Hz to observe. The repetition rate of pulse mode, 10 Hz, made it impossible to observe 50 Hz behaviour but CW mode with a high framerate camera, in this case 240 Hz, allows for a much greater frequency range to be observed. Focusing the CW electron beam onto the detector gives high enough intensity to resolve the beam centre with only 4 ms camera exposure.

In order to examine the temporal behaviour of the beam drift a faster camera was used to

observe the MCP[6] with the electron beam focused at the detector. To operate the camera at the desired framerate it was necessary to reduce the size of the image from the camera to a smaller field of view. Data was collected with the CAES operating in continuous mode and in pulsed mode.

To assign accurate timestamps to each image it was necessary to use a feature of the camera that adds an accurate truncated timestamp directly to the image combined with the inaccurate timestamp on the image file created by the computer the camera was running on. The combination of the fine and accurate timestamp from the camera (which only counted to 128 seconds before resetting) with the coarse inaccurate timestamp on the image file allowed for long duration data sets to be accurately analysed.

To examine the temporal behaviour of the source stability the beam position coordinates were deduced from the images and then Fourier transformed. With the source operating in continuous mode the analysis was limited by the framerate of the camera (240 Hz) and in pulsed mode it was limited by the repetition rate of the experiment (10 Hz). Shown in Figure 2.11 are the recorded positions of the electron beam and the corresponding Fourier transforms for continuous and pulsed mode. With the low sampling rate of the pulsed mode there is little information in the Fourier transform with the only feature being a peak near 0 Hz corresponding to the slow oscillation visible in the data. The higher sampling rate of the continuous mode data provides much more information with numerous peaks visible in the spectrum. The continuous mode spectrum reveals a strong contribution at 50 Hz, likely related to the mains power supply, as well at peaks at 45 Hz, 67 Hz, 84 Hz, and 92 Hz which are of unknown origin.

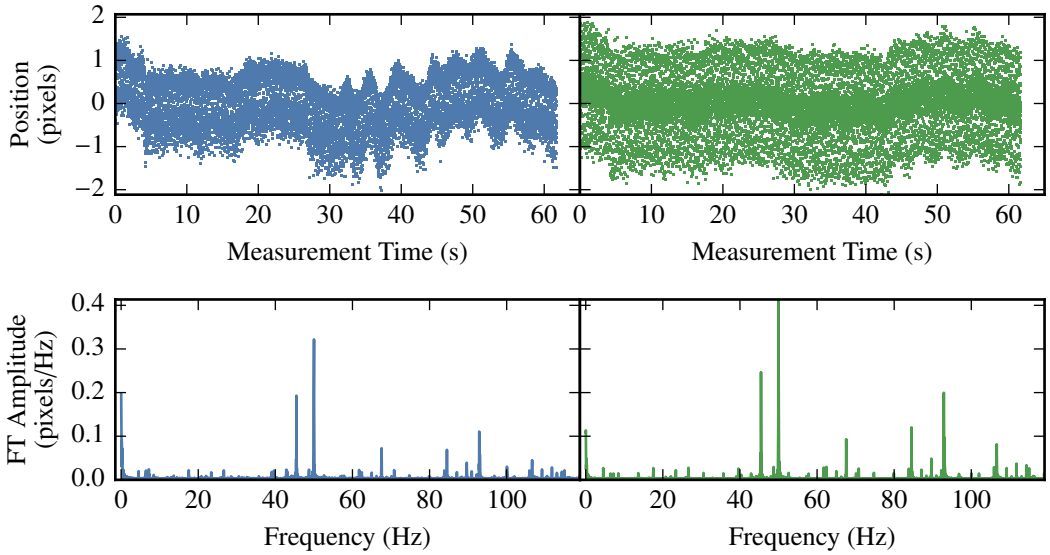
The instabilities present in the normal pulsed operation are still apparent in the continuous mode beam indicating that the fast switching of the magnetic-fields was not the primary source of the beam drift. The 50 Hz peaks in the position spectrum are indicative of some interaction with the main power supply however care had already been taken to keep power supplies and similar devices as far from the beamline as was practical.

2.2.4 Summary

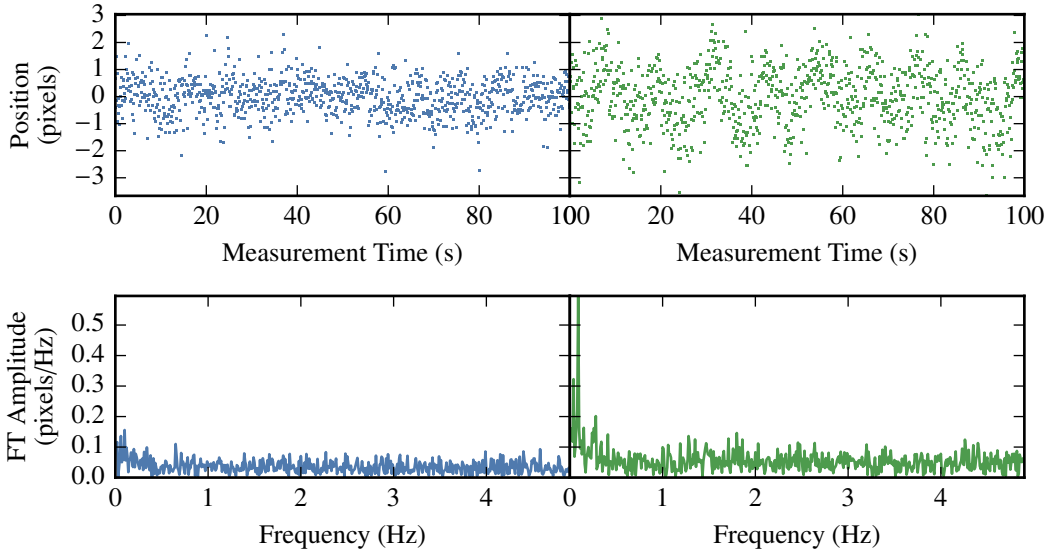
It was hoped that continuous mode would provide a viable alternative to normal pulsed operation with this iteration of the CAEIS with better beam current and stability. An increase in the time-averaged beam current of approximately a factor of six is possible, compared to pulsed mode, by increasing the atomic flux and intensity of the blue ionisation laser. The fast switching on and off of the MOT and Zeeman slower magnetic coils were eliminated as a primary candidate for the electron beam instabilities as the magnitude of the drift was more or less the same with pulsed and CW mode.

The investigations in this section indicate that, with the current apparatus, continuous mode is able to provide an increase in beam current but no increase in beam stability at the cost of

[6]Point Grey Flea3 FL3-U3-13S2M-CS



(a) Continuous electron beam.



(b) Pulsed electron beam.

Figure 2.11: Beam position and position spectrum for x (blue) and y (green) directions for CW-mode electrons (a) and pulse-mode electrons (b). Pulsed operation is limited to 10 Hz and thus the spectrum is limited to 5 Hz. The continuous beam spectrum was limited to 120 Hz by the camera framerate of 240 Hz.

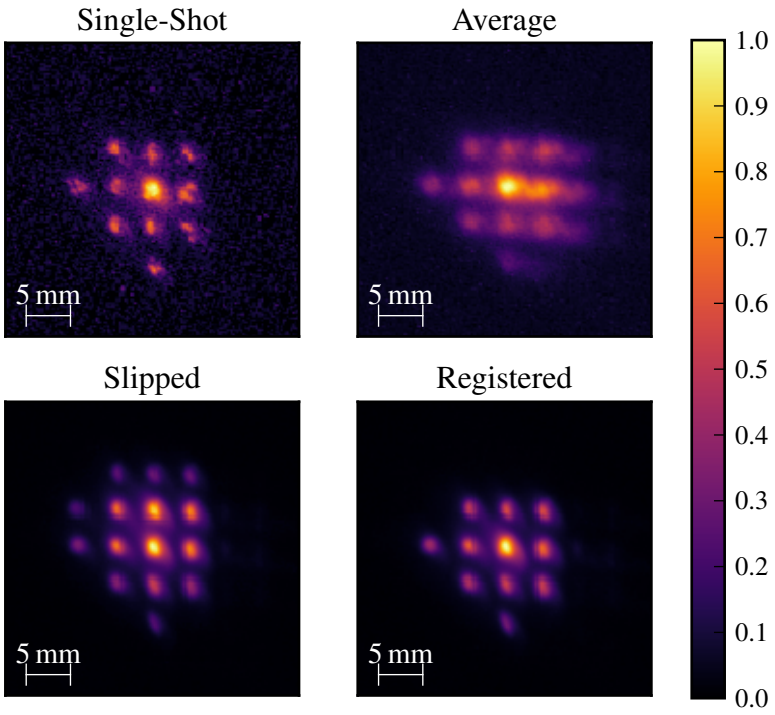


Figure 2.12: A demonstration of the importance of the image registration technique using a contrived example of slipping. *Single-Shot* is a single image from a prototype-pepperpot data set of 1000 images. *Average* is the simple average of all 1000 images. *Slipped* is a contrived demonstration of slipping; there is an additional set of beamlets visible at the top of the image. *Registration* is a successful registration where a shot-to-shot limit to drift was enforced. All the images are linearly scaled as indicated by the colour bar.

no longer being able to produce ultrafast bunches. The next generation of the CAEIS now under construction will operate as a continuous ion source, capitalising on the beam current achievable by directly ionising the atom beam [79].

2.3 Registration

Instability in the beam path (see Section 2.2.3) resulted in the electron beam trajectory drifting from shot-to-shot. When multi-shot averages were required the beam drift resulted in the beam being 'smeared' when averaged. It was necessary to 'register' the images to prevent this smearing and this is demonstrated in Figures 2.12 and 2.13.

Registration consisted of convolving each image with a reference image, recording the location of the maximum value of the convolution and then using that location to align all the images. The reference image usually consisted of either just the first image in the set or the

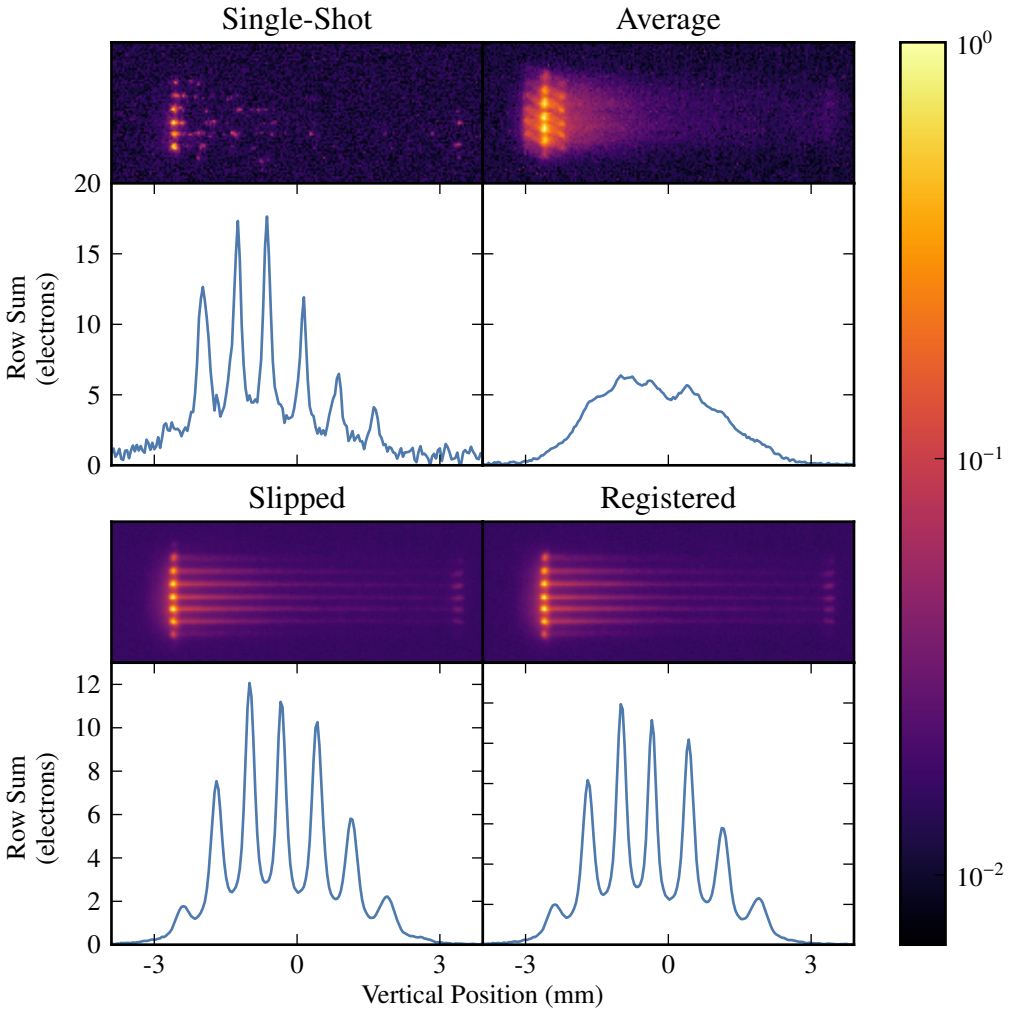


Figure 2.13: A demonstration of the image registration technique using a real example. *Single-Shot* is a single image from a streaked pepperpot data set of 1000 images. *Average* is the simple average of all 1000 images. *Slipped* is an attempt at registration that only looks for the maximum value of the convolution with no limit on shot-to-shot variation; there is an additional faint beamlet visible at the top of the image. *Registration* is a successful registration where a shot-to-shot limit to drift was enforced. Below each image is the row sum of that image. All the images are log scaled as indicated by the colour bar.

average image of the entire set, depending on the amount of signal in a single image. The registration process could also be performed iteratively where the reference image became the aligned output of the previous iteration.

Registration functioned very well for low noise, structured data such as diffraction patterns (see Chapter 4) or pepperpot beam masks (see Chapter 5). Low signal data could be registered as long as there was sufficient signal in a single shot for the convolution to correctly identify the features of the image. Unstructured data or continuous data, such as large flat beam profiles, did not register well as the convolution was not able to identify common features well.

Noisier data required more care than clean data and in some cases could not be registered. One of the registration errors that has occurred with pepperpot data is 'slipping' where the convolution maxima correspond to neighbouring beamlets rather than the same beamlet and thus the images are incorrectly aligned. This error could be evaded by applying a limit to how far the convolution maxima of an image could change from shot to shot, generally the limit should be less than the distance between beamlets. This allowed slow drift of the beam path to be accounted for while preventing slipping.

Slipped, and therefore failed, pepperpot registrations were usually easy to identify since the number of beamlets was known and slipped errors results in extra beamlets appearing in the registered image. While registering the pepperpots the best results were provided with around 3 iterations and limits on the shot-to-shot drift less than distance between the pepperpots beamlets. An contrived example of a slipped registration is shown in Figure 2.12 and a real, albeit subtle, example can be seen in Figure 2.13 using data from Chapter 5.

2.4 Astigmatism

During the investigations already discussed in this section, astigmatism was observed in the electron beam. The astigmatism was apparent when the strength of the electron lenses were adjusted or while translating the MCP detector past a beam focus as the two orthogonal transverse axes did not share a focal point, as shown in Figures 2.14 and 2.18a.

A quadrupole magnetic field can be used to correct for astigmatism in a beam along a fixed axis. A quadrupole magnetic field can be described as

$$\begin{aligned} B_x &= Ky \\ B_y &= Kx, \end{aligned} \tag{2.2}$$

where the constant K indicates the strength of the field, and x and y refer to the transverse displacement from the central axis. The force on particles in a charged beam, such as that in the CAEIS, passing through a quadrupole field is shown in Figure 2.15 where there is an inwards force along one axis and an outwards force along the other, which is ideal for correcting astigmatism.

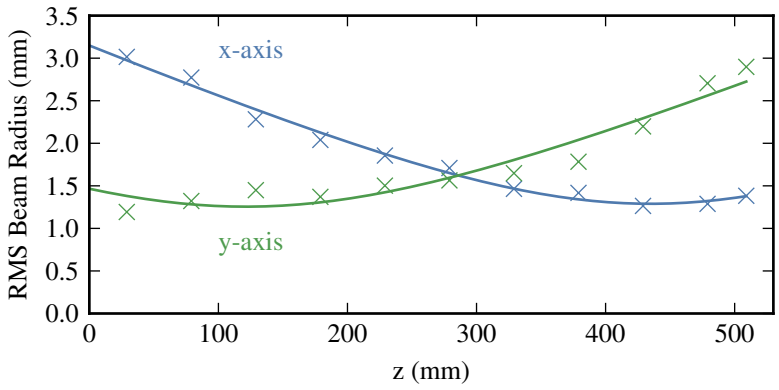
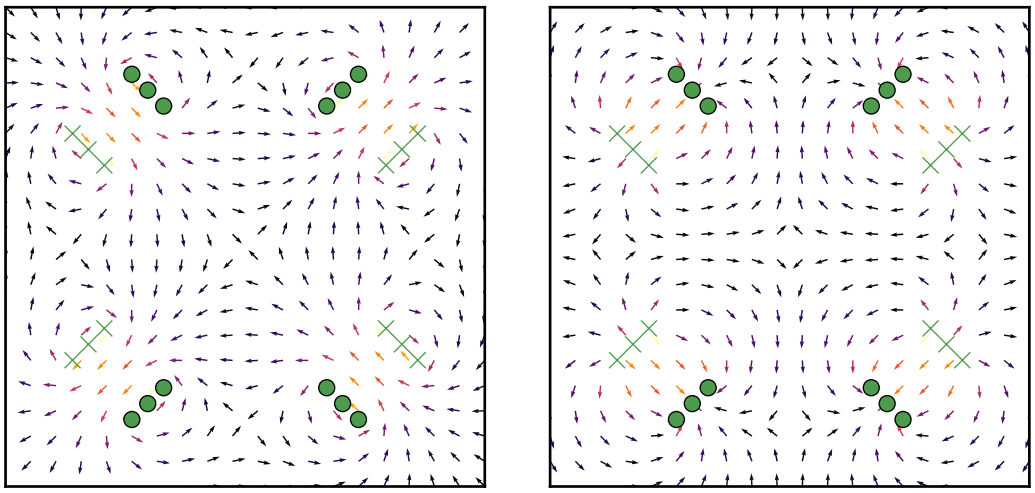


Figure 2.14: A demonstration of astigmatism in the electron beam as the detector is translated past the focal points. The x- (blue) and y-axis (green) root mean square (RMS) beam radii are shown on the vertical figure axis and the horizontal figure axis shows the MCP detector position along the beam axis, z. The crosses indicate the measured beam size and the lines a fit to the Gaussian beam waist equation. The beam astigmatism is apparent as the two axes do not come to a focus at the same position.



(a) Magnetic Field

(b) Force Field

Figure 2.15: The relative magnetic field and force on a moving electron for a quadrupole lens. (a) shows the magnetic field, and (b) shows the force field for an electron beam travelling out of the page, (b), where the arrows indicate the direction and strength of the field. The green circles and crosses indicated the cross section of the quadrupole solenoids with circles representing current travelling out of the page and crosses into the page.

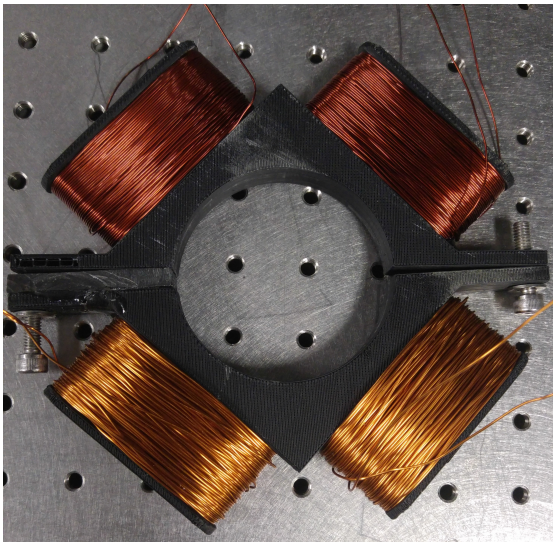


Figure 2.16: The 3D printed chassis wound with copper wire to form a quadrupole lens. The scale is apparent from 70 mm inner diameter of the lens and the 25 mm spacing of the holes of the optical bench the lens is resting on.

To correct the astigmatism in the CAES a simple solenoid magnetic quadrupole lens was designed and built consisting of four solenoid magnets, orientated radially, and spaced around the beamline with alternating current directions as shown in Figures 2.15 and 2.16. The design was constrained as it had to be external to the vacuum system with an inner diameter of 70 mm and have a longitudinal length less than 20 mm due to the crowded beamline. Simulations were performed to determine the approximate number of current turns required for the solenoids forming the quadrupole lens as well as the ideal transverse solenoid length.

2.4.1 Quadrupole Simulations

A simple charged particle simulator was created with the ability to propagate charged particles through arbitrary magnetic fields which in this case were formed by a parameterised quadrupole lens. The code for these simulations is given in Appendix B. The electrons used in the simulations had a beam energy of 8.5 keV and an initial RMS beam radius of 5 mm, similar to electron bunches produced by the CAES. The electrons were given normally distributed randomly determined initial transverse velocities with the x velocity distribution given an RMS width of $10\,\mathrm{km\,s^{-1}}$ and the y velocity distribution given an RMS width of $20\,\mathrm{km\,s^{-1}}$ which produced an astigmatism similar to that observed experimentally. The electrons bunches were propagated 250 mm to the virtual quadrupole lens, through the lens, for another 250 mm to a focusing lens, and finally for another 500 mm to observe the beam foci. A number of figures of merit were available to determine the ideal design, such as the degree to which the quadrupole lens was able to correct the astigmatism, the current required for the lens to correct for the astigmatism, and the size of the beam waist.

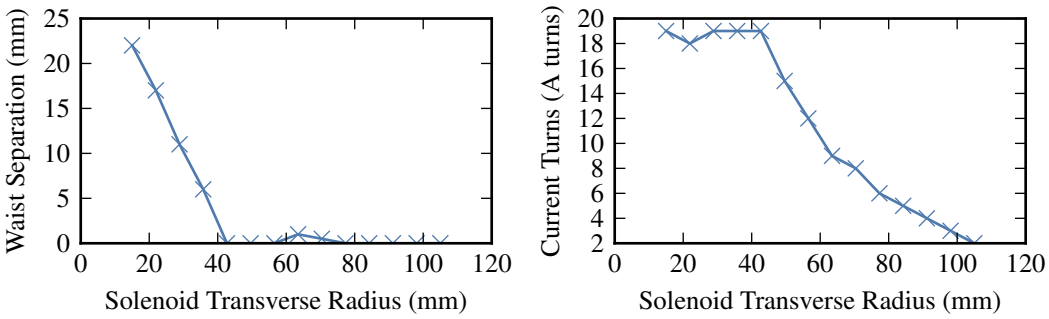


Figure 2.17: Particle simulation results from propagating 8.5 keV electrons through a quadrupole lens while varying the quadrupole parameters. The electrons had an initial RMS bunch radius of 5 mm and were given initial velocities to replicate the astigmatism observed in the CAES. The left figure shows the minimum separation of the beams waists in the x and y direction as the transverse radius of the solenoids forming the quadrupole was varied. The right figure shows the current turns required to achieve the minimum waist separation as the transverse solenoid radius was varied.

A portion of the simulation results are shown in Figure 2.17 and the main figures of merit used were the separation of the foci for each axis and the number of current turns required to achieve minimum foci separation for a given solenoid transverse radius. The results indicate that negligible waist separation requires a transverse solenoid radius of at least 35 mm and that the larger the transverse radius the lower the required current turns through the solenoids.

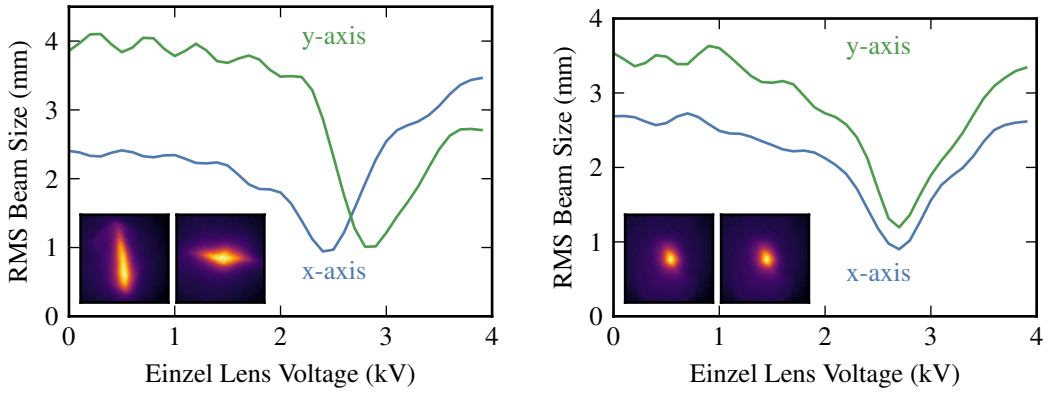
2.4.2 Quadrupole Construction

The selection of parameters for the construction of the quadrupole lens was informed by the simulations and a transverse radius of 35 mm was chosen to provide small beam waist separation with a manageable number of current turns given the current supplies available. Another consideration was the space available along the beamline: if a wider solenoid transverse width was chosen then there would be additional complexity required to mount the lens to the vacuum system in the limited space available.

The chassis for the quadrupole solenoids was modelled and 3D printed from plastic and the solenoid coils were hand wound onto the chassis, as shown in Figure 2.16. Each of the solenoid coils had approximately 100 turns.

2.4.3 Quadrupole Correction

The quadrupole lens functioned as desired and was able to correct the astigmatism as demonstrated in Figure 2.18. The lens required a current of approximately 1 A running through the solenoids in order to correct the astigmatism. The lens was used during a number of measure-



(a) Quadrupole Off

(b) Quadrupole On

Figure 2.18: The size of the electron beam on the detector as the Einzel lens voltage is varied with the quadrupole lens off (a) and on (b). The blue lines indicate the horizontal, x, RMS beam size and the green lines the vertical, y, RMS beam size. Inset in each figure are linearly scaled images of the beam profiles for minimum spot size for each axis with colour scales as indicated in Figure 2.12. The quadrupole is able to correct the astigmatism present in the beam.

ments to improve the bunch quality; an example of this is shown in Figure 5.5.

2.5 Summary

The University of Melbourne CAEIS has undergone many changes and improvements since its inception, the most recent of which have been detailed in this chapter. The apparatus was able to produce low temperature electron and ion bunches with arbitrary transverse profiles which had sufficient coherence for the imaging of small biomolecules.

Careful manipulation of the ionisation mechanisms of CAEIS allowed for the production of electron and ion bunches with duration that ranged from 10s of picoseconds to nanoseconds to 10s of microseconds. Picosecond duration bunches have the potential to be used with bunch compression to produce femtosecond duration bunches for UED. The remaining obstacles to swift progress towards the production of molecular movies are the low beam current, particularly for ultrafast bunches, and the unstable electron trajectories.

A comparison of the usual pulsed CAEIS operation with continuous beam production was undertaken in the hopes that beam current and electron trajectory stability could be improved. This investigation did not result in an improvement to beam stability but did prove useful in eliminating the MOT coils as a potential cause and identifying the 50 Hz frequency of the primary noise source. Continuous operation was able to produce higher current beams than pulsed operation.

The registration algorithm is an essential component of the CAEIS that deals with electron

beam path instability by aligning images from multiple shots as the beam drifts. This technique allows for measurement that would otherwise not be possible and is essential to the diffraction and brightness measurements that will be presented in Chapters 4 and 5.

Beam astigmatism in the electron beam was also detected and characterised. A magnetic quadrupole lens was designed and built to correct this astigmatism. This quadrupole lens was used to improve the beam quality for the measurements presented in Chapter 5.

Chapter 3

Laser Frequency Stabilisation

Laser frequency stabilisation is an essential tool for many atomic physics experiments [80–85]. There are a plethora of techniques available for laser frequency stabilisation each with numerous advantages and disadvantages. Of particular interest here are stabilisation techniques that utilise high-bandwidth feedback to produce laser sources with narrow spectral linewidth which are important to applications such as atomic clocks [86], high-resolution spectroscopy [87], and metrology [73, 88].

Laser frequency stabilisation is an essential component of the CAEIS as it is required for the MOT to function, for imaging of the atomic cloud and for the precise control involved in the ionisation process. Relatively simple techniques such as saturated absorption spectroscopy [89–91] are adequate for the frequency linewidths required for the MOT. More precise methods are useful when interacting with the Rydberg states of an atom as some of the Rydberg transitions have very narrow linewidths.

PDH locking with a high-finesse cavity [92] is a proposed method for precise control over laser frequency. PDH has been used to produce extremely good frequency stabilisation with sub-40 mHz linewidths achievable and is an essential part of the frequency stabilisation systems used at the Laser Interferometer Gravitational-Wave Observatory (LIGO) [93–96]. The PDH technique is unfortunately not relative to an absolute frequency reference, such as an atomic transition. In order to capitalise on the narrow linewidth achievable with PDH locking while ensuring absolute frequency stability, PDH can be combined with saturation or polarisation spectroscopy to an absolute frequency reference such as an atomic transition, to prevent slower frequency drifts due to changes in the optical cavity resonance frequency as temperature and pressure changes in the lab.

The focus of this chapter is on polarisation spectroscopy (PS) which was first described by Wieman and Hänch in 1976 as, "...a sensitive new method of Doppler-free spectroscopy, monitoring the nonlinear interaction of two monochromatic laser beams in an absorbing gas via changes in light polarisation." [51, 97]. It has been shown previously that PS can be used to reduce the linewidth of a distributed feedback diode from 2 MHz to 20 kHz [98] and of an

ECDL to 65 kHz [99]. During the course of the research described here, PS was demonstrated to be capable of linewidth reduction to sub-kilohertz linewidth if high-bandwidth feedback is used. This work has been published as Reference [52].

This chapter provides an overview of laser frequency stabilisation, a discussion of the physics of PS followed by details on the implementation and measurement of high bandwidth frequency stabilisation using PS. Much of the work described in this chapter was conducted as part of an academic/industrial collaboration project with industry partner MOG Laboratories Pty. Ltd. (”MOGLabs”). The research outcomes were instrumental in improving the bandwidth of the electronics used within MOGLabs lasers which have since been used to achieve sub-hertz linewidths in a commercial system [100].

3.1 Laser Frequency Stabilisation

Laser frequency stabilisation is used to reduce the frequency spread of a laser. Laser frequency stabilisation can range from weak stabilisation keeping the centre frequency of a laser at a particular frequency to convoluted frequency narrowing techniques that attempt to reduce laser spectral linewidth to sub-hertz levels. These techniques use a frequency reference such as an optical cavity or atomic transition and provide negative feedback to the laser, using a servo system, to keep the laser at the reference frequency.

A frequency discrimination method is used to generate an error signal and a servo system drives various feedback actuators to minimise that error. A servo system is a system that uses error sensing negative feedback to control a device via an actuator. A simple example of a servo is the cruise control in a car where the difference between the desired speed and the actual speed (the error signal) is used to modulate the throttle to get closer to the desired speed. Laser stabilisation systems use servo systems to appropriately apply gain to the error signal and apply the result to the various feedback actuators available.

The efficacy of stabilisation techniques can be described by the width of the frequency distribution of the laser, called the linewidth. Linewidth usually refers to either the FWHM or RMS spectral width about the central frequency and is used to describe measurements made over various timescales. Short duration measurements, usually less than a second, are used to describe the linewidth of lasers whereas long timescale measurement, hours or days in duration, are used to describe the drift of laser central frequency over time.

A number of traits are desirable in a laser frequency stabilisation scheme including:

- *Absolute frequency reference*: Frequency stabilisation techniques that rely on optical cavities are vulnerable to changes in the cavity resonant frequency, due to changes in cavity length with temperature or pressure for example, whereas other techniques such as saturated absorption spectroscopy (SA) or PS are relative to atomic energy levels and thus not subject to drift in the same way.

- *High-bandwidth*: All else being equal, high-bandwidth techniques provide greater potential for linewidth reduction than low-bandwidth techniques.
- *Modulation-free*: A number of stabilisation techniques require frequency- or phase-modulation which limits the bandwidth of the technique to half the modulation frequency due to the Nyquist limit; thus modulation-free, or high-frequency modulation techniques are often preferable. Modulation-free techniques are often more susceptible to low frequency noise than modulated techniques.
- *Low drift*: Drift can occur with slow changes to the lock point of the stabilisation scheme, potentially due to ambient changes in lab temperature or pressure, or the electrical environment causing subsequent changes to laser beam power, polarisation or locking electronics voltage levels all of which can result in drift in the laser frequency. This can even occur with techniques that use an absolute frequency reference although non-absolute techniques tend to be more susceptible to drift.
- *Stable*: Some techniques are more susceptible than others to unlocking, where perturbations such as ambient temperature or pressure changes, or percussive events (doors closing, dropped tools) cause large sudden changes in laser frequency which the servo system is unable to compensate for. Techniques with large capture ranges tend to have greater stability due to being able recover from larger frequency perturbations, see Section 3.4.1 for an example.

There are a large number of available techniques and variations on techniques for stabilisation each with different advantages and drawbacks. A few of these techniques are SA [89–91, 101, 102], dichroic atomic vapour laser lock (DAVLL) [103, 104], modulation transfer spectroscopy (MTS) [105–108], Sagnac interferometry [109, 110], polarisation spectroscopy (PS) [51, 98, 99, 111–115], PDH [92], and Hänsch Couillaud stabilisation [116].

3.1.1 Frequency Control and Feedback

A number of methods can be used to control the output frequency of a laser and these can be used in concert to supply feedback from the frequency reference in order to stabilise the laser frequency and decrease the spectral linewidth. The focus here will be on the feedback systems of diode lasers, particularly those of an ECDL.

Temperature

The temperature of a laser diode affects the output frequency due to the temperature dependence of the optical path length, gain curves and the thermal expansion of the external cavity with an ECDL [117]. The processes affect output frequency at different rates and all show an increase in wavelength with increasing temperature.

The temperature of laser diodes can be controlled through two methods. A thermo-electric cooler (TEC) can be used to directly control the temperature of the device, and the injection current into the diode affects the temperature through resistive heating effects. Good insulation and thermal inertia also contribute to the stability of diode temperature [66].

Temperature is not typically used to directly manipulate the frequency of a laser due to the relatively slow response as changes in the control signal take seconds to propagate from the TEC to the diode and longer to fully thermalise. Typically temperature is just stabilised to reduce the impact of ambient temperature changes on the performance of the diode laser.

Injection Current

Modulation of the injection current into the laser diode is one of most common ways of controlling the output wavelength of a diode laser. The injection current into the diode affects the temperature of the diode, and the density of charge carriers which in turn affects the refractive index of the medium and thus the wavelength of the laser light produced.

Modulation of the injection current is the fastest feedback method available to diode lasers and is able to suppress noise up to MHz ranges [52, 93]. The design of the electronics involved in the modulation of the injection current can have a noticeable influence on the performance at high frequencies as noise becomes an issue.

The lasers used in the experiments described in this thesis all had two modes of current feedback, 'slow' current feedback, with bandwidth from DC to approximately 100 kHz, and 'fast' current feedback, with bandwidth from about 100 kHz to 50 MHz.

The investigations of laser spectral behaviour, discussed later in this chapter, were instrumental in identifying potential improvements to the design of the electronics involved in the fast current channel of MOGLabs diode lasers. The measurements presented were made with the final prototype of the enhanced laser headboard, now standard with lasers made by MOGLabs.

Grating Angle and Position

In external cavity diode lasers, for example the Littrow configuration ECDL shown in Figure 3.1, the angle and position of the external grating can be used to control the output frequency. The grating angle affects the wavelength of the light that is reflected back into the laser diode from the second-order reflection and thus the angle can be used to select output wavelength. The grating position is used to control the length of the external cavity which also determines the laser wavelength and is commonly controlled using piezoelectric actuators [118].

Wavelength control via the grating angle is limited by the response rate of the piezo actuator which tend to respond in microseconds. Thus the grating angle can be used to deal with relatively low frequency noise, up to approximately 1 kHz or so, such as changes to ambient

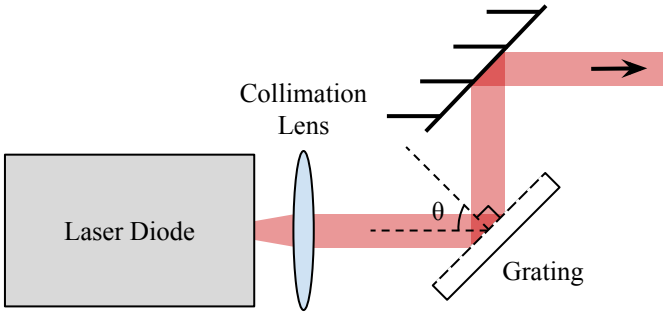


Figure 3.1: Littrow configuration for diode lasers. The raw output of the diode is collimated and then incident on an optical grating. The angle of the grating, θ, changes what frequency of light is coupled back into the diode from the first-order reflection and thus the frequency of output.

temperature, pressure, or some acoustic vibrations.

3.1.2 Noise

In this context noise refers to effects that change the frequency of a laser in undesirable ways. Some sources of noise are thermal changes, ambient vibrations, atmospheric pressure, or the electrical power supply.

Thermal noise can be caused by a number of effects such as changes in weather or unreliable building climate control. Thermal noise can directly affect the length of the laser cavity and the refractive index of air within the cavity but also affects the alignment of optics, the efficiency and polarisation of light transmitted through fibres, and atomic vapour cell opacity which can erroneously be interpreted by some frequency stabilisation systems as frequency changes and 'corrected'.

Electrical noise can occur with changes to the wider electrical grid as well as when devices in the lab are turned on or off, fast switching high-voltage/-current supplies, such as those used to switch the MOT coils, can cause noticeable effects on other electrical equipment in close proximity. Noise in the electronic environment can cause frequency instability particularly if a laser diode is not fully isolated from the electrical ground. Noise on the power supply to the laser diode affects the power and frequency of the light emitted. Laser intensity noise can also cause problems, for example laser frequency stabilisation is often conducted with spectroscopic techniques that will interpret intensity fluctuations as frequency fluctuations. Thus the feedback system will create frequency noise as it attempts to correct for the phantom frequency noise.

Mechanical noise can have numerous sources such as audible noise, percussive noise (dropped spanners, doors) or vibrational sources such as cooling fans on nearby equipment. An ECDL subjected to mechanical vibrations will experience frequency noise as the diode cavity length and refractive index varies. Mechanical noise can also affect the alignment, and thus transmit-

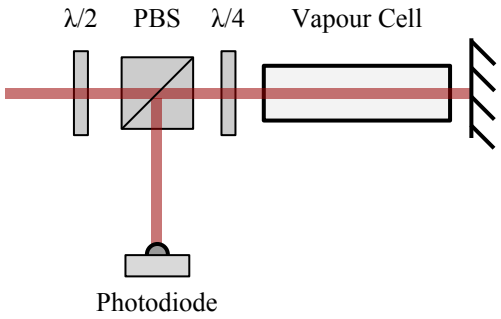


Figure 3.2: An example of a saturated absorption spectroscopy setup. λ/2 and λ/4 refer to half- and quarter wave phase retarders respectively and are used to control the intensity of the light travelling through the vapour cell and hitting the detector using the polarising beam splitter (PBS).

ted power, of light through optical fibres, optical isolators and apertures, which in turn can be falsely interpreted by the servo system as a change in frequency and 'corrected'.

3.2 Saturated Absorption Spectroscopy

Saturated absorption spectroscopy (SA) is a simple and common technique for laser frequency stabilisation which can be used with a number of atomic species and is a staple of atom optics laboratories [97]. SA is often used in applications where extremely narrow linewidths are not required due to the relative simplicity of the method such as MOT trapping and cooling, or atom cloud imaging. A schematic of SA is shown in Figure 3.2.

SA involves counter-propagating pump and probe beams from the laser source through a sample of atomic vapour with the intensity of the probe beam after the gas measured by a photodetector. Without the pump beam the absorption of the probe as a function of laser frequency would show peaks at the atomic transitions. The width of the absorption peaks is equal to the linewidth of each transition, broadened by the thermal distribution of the atoms due to thermal motion. At room temperature the absorption spectrum for rubidium is a smooth curve 100s of MHz wide which obscures the hyperfine transitions. Adding the pump beam results in less absorption at each transition due to the pump exciting atoms which are then inaccessible to the probe. With counter-propagating beams 'cross-over' features appear halfway between each pair of transitions [97]. An example spectrum for rubidium-85 is shown in Figure 3.3. In Figure 3.2 the pump beam is recycled to act as the probe.

Different variations of SA can operate with or without frequency modulation, which can be modulation of the laser (at the laser diode or just the beam with an AOM or electro-optic modulator (EOM)) or modulation of the transition frequencies of the atoms in the vapour cell (via a solenoid magnet wrapped around the vapour cell). Without modulation the system is

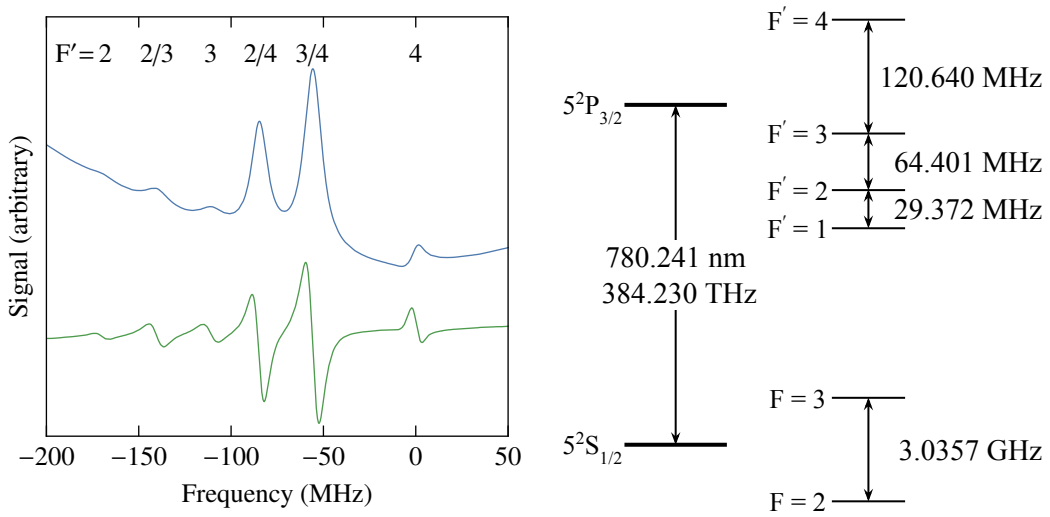


Figure 3.3: An example of a saturated absorption spectroscopy absorption spectrum for the rubidium-85 D2 transition. The blue line shows the absorption spectrum and the green line shows the error signal for modulated SA. From left to right the peaks in the absorption spectrum are the $F = 3$ to $F' = 2, 2/3, 3, 2/4, 3/4$, and 4, where x/y refers to a crossover transition. The figure on the right shows the energy levels of the rubidium-85 D2 transition [119].

more susceptible to drift and there is a frequency offset from the centre of the atomic transition. If modulation is used then there is the added complexity of implementing the modulation and the feedback bandwidth is limited to half the modulation frequency. Laser linewidth under 150 kHz is attainable with modulated saturated absorption spectroscopy [102].

3.3 Pound Drever Hall

The Pound-Drever-Hall (PDH) technique is the gold-standard for laser frequency linewidth reduction and has been used to achieve extremely low linewidth of less than 40 mHz [94]. PDH uses an optical cavity as a frequency reference and a modulated beam and some electronics to generate the error signal for feedback to the laser [92, 96]. PDH is a phase sensitive measurement that compares the laser frequency with the light stored in the optical cavity and thus is not bandwidth limited by the spontaneous emission lifetime. A schematic of a standard PDH setup is shown in Figure 3.4.

PDH was used to provide a comparison to PS and the optical cavity was used extensively as a diagnostic tool for examining laser frequency behaviour.

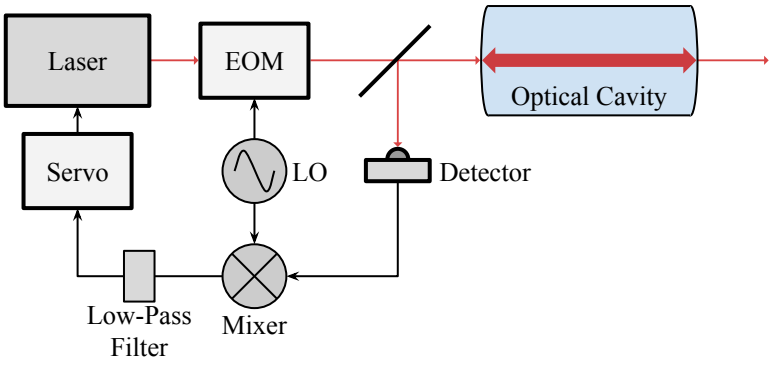


Figure 3.4: A standard PDH layout. The laser beam passes through an EOM to create frequency sidebands and is then incident on the optical cavity. The beam reflected from the cavity is detected with a photodetector and the signal passed through appropriate electronics to produce the error signal (see Section 3.3.1). The error signal is passed to the servo system to generate feedback to keep the laser frequency on the cavity resonance.

3.3.1 Fabry-Pérot Cavities

A Fabry-Pérot cavity is formed by two highly reflective mirrors facing each other such that light can form a standing wave between the two mirrors. Laser light incident on a Fabry-Pérot cavity will only couple into the cavity if the length of the cavity is equal to an integer number of wavelengths of the light. So for an ideal laser to be resonant with an ideal cavity,

$$L = n\lambda, \tag{3.1}$$

where L is the length of the cavity, λ is the wavelength of the laser and n is some integer. Realistic lasers and cavities have finite linewidths which when viewed as a transmission or reflection spectrum are shown as a convolution of the cavity and laser linewidth.

The frequency difference between one cavity transmission and the next is called the free-spectral range (FSR), $\Delta\nu_{FSR}$ and depends on the length of the cavity,

$$\Delta\nu_{FSR} = \frac{c}{2L}$$

The quality of an optical cavity is described by the cavity finesse, $\mathscr{F}$, which effectively describes the number of traversals a beam makes before leaking out or being absorbed and it is determined by the intensity reflectivity of the mirrors, R, [120]

$$\mathscr{F} = \pi\frac{\sqrt{R}}{1-R}.$$

The light transmitted through an optical cavity can be described by [121]

$$T = \frac{1}{1 + F\sin^2\frac{d}{2}}, \tag{3.2}$$

where $d = 2\pi\frac{2L}{\lambda}$ and $F = \frac{4R}{(1-R)^2}$ is the coefficient of finesse. The coefficient of finesse is related to the finesse by

$$\mathscr{F} = \frac{\pi}{2}\sqrt{F}. \tag{3.3}$$

The phase difference between successive traversals of the cavity is represented by d. We can define the difference between the laser wavelength and closest cavity resonance as $\Delta\lambda = L - n\lambda$, then Equation 3.2 can be written as

$$\begin{aligned} T &= \frac{1}{1 + F\sin^2\left(2\pi\frac{\Delta\lambda}{\lambda}\right)}. \\ &\simeq \frac{1}{1 + F\left(\frac{\omega L}{c}\right)^2}, \end{aligned} \tag{3.4}$$

where ω is the angular frequency of the laser. T takes the form of a Airy function and an example spectra is shown in Figure 3.5.

We used a high-finesse cavity from Stable Laser Systems, isolated inside a temperature controlled vacuum chamber. The cavity was a hemispherical cavity constructed from ultra low expansion glass where the curved mirror had a radius of curvature of 0.500 m and both mirrors were coated for high reflectivity for 780 nm and 960 nm light. The cavity was 10 cm long, had a finesse of 20942, mirrors with a reflectivity of 0.99985, an FSR of 1.5 GHz, and a FWHM linewidth of 71.6 kHz. This cavity had fixed mirrors unlike some others that have one of the mirrors attached to a piezoactuator to allow for modulation of the cavity resonance.

PDH Error Signal

The magnitude of the optical electric field incident on a Fabry-Pérot cavity, if the frequency is assumed to be approximately constant, can be written as

$$E_I = E_0 e^{i\omega t}, \tag{3.5}$$

where $\omega = 2\pi f_l$ is the angular frequency of the laser. The light reflected from the cavity consists of the reflected beam and the leakage beam from light that has traversed the cavity one or more times before leaking out of the first mirror. The reflected beam undergoes a phase shift of π relative to the incident beam. The leakage beam has numerous phase components with one cavity round trip giving a phase shift of $-2L\omega/c$, two round trips giving $-4L\omega/c$, and so on. Thus the reflected beam electric field will be

$$E_R = E_0\left(re^{i(\omega t+\pi)} + trte^{i(\omega t-2L\omega/c)} + tr^3te^{i(\omega t-4L\omega/c)} + ...\right), \tag{3.6}$$

where r is the mirror reflectivity and $t = \sqrt{1-r^2}$ is the transmissivity of the mirrors. Equation 3.6 can be simplified to give the reflection coefficient

$$F_R(\omega) \equiv \frac{E_R}{E_I} = \frac{r\left(e^{i\omega/\Delta\nu_{FSR}} - 1\right)}{1 - r^2 e^{i\omega/\Delta\nu_{FSR}}}. \tag{3.7}$$

Figure 3.5 shows that $F_R(\omega)$ is antisymmetric about the cavity resonance making it ideal for frequency stabilisation however more steps are required to generate a usable error signal.

To generate the error signal phase modulation is imposed on the laser beam, typically using an EOM. The modulated incident laser beam with a modulation frequency of Ω and modulation strength β has electric field

$$\begin{aligned} E_I &= E_0 e^{i(\omega t + \beta \sin \Omega t)} \\ &\approx E_0\Big(J_0(\beta) + 2iJ_1(\beta)\sin\Omega t\Big)e^{i\omega t} \\ &= E_0\Big(J_0(\beta)e^{i\omega t} + J_1(\beta)e^{i(\omega+\Omega)t} - J_1(\beta)e^{i(\omega-\Omega)t}\Big), \end{aligned} \tag{3.8}$$

valid for small β where J_0 and J_1 are Bessel functions. Equation 3.8 shows that the phase modulated beam has three frequency components and the two J_1 components are referred to as the sidebands which are offset from the central frequency by the modulation frequency Ω.

The reflected beam from the cavity due to the phase-modulated beam results in each frequency component being transformed by $F_R(\omega)$ to give

$$E_R = E_0\Big(F_R(\omega)J_0(\beta)e^{i\omega t} + F_R(\omega+\Omega)J_1(\beta)e^{i(\omega+\Omega)t} - F_R(\omega-\Omega)J_1(\beta)e^{i(\omega-\Omega)t}\Big). \tag{3.9}$$

The electric field is not directly measured, photodetectors measure the power, $P = |E|^2$, of the beam. Thus,

$$\begin{aligned} P_R \quad = \quad & P_c|F_R(\omega)|^2 \quad + \quad P_s\Big(F_R(\omega+\Omega) + F_R(\omega-\Omega)\Big) \quad + \\ & 2\sqrt{P_cP_s}\ \mathrm{Re}\Big[F_R(\omega)F_R^*(\omega+\Omega) - F_R^*(\omega)F_R(\omega-\Omega)\Big]\cdot\cos\Omega t \quad + \\ & 2\sqrt{P_cP_s}\ \mathrm{Im}\Big[F_R(\omega)F_R^*(\omega+\Omega) - F_R^*(\omega)F_R(\omega-\Omega)\Big]\cdot\sin\Omega t \quad + \quad O[2\Omega], \end{aligned} \tag{3.10}$$

where $P_{c,s}$ are the power of the carrier and sideband components respectively and the final term represents the higher-order components from the interactions between the sidebands.

The phase information is retrieved with the use of a "mixer" which is an electronic device that multiplies two signals together essentially multiplying the signal from the photodetector, $P_R \propto \sin\Omega t$, with the signal from the oscillator driving the EOM, $\sin\Omega t$. The oscillating portions of the third and forth terms in Equation 3.10 when mixed become

$$\begin{aligned} \cos\Omega t \sin\Omega t &= \frac{1}{2}\sin 2\Omega \\ \sin\Omega t \sin\Omega t &= \frac{1-\cos 2\Omega t}{2}, \end{aligned} \tag{3.11}$$

thus resulting in a DC component and 2Ω components. A low-pass electronic filter is then used to extract the DC component which forms the PDH error signal,

$$\epsilon = 2\sqrt{P_cP_s}\ \mathrm{Im}\Big[F_R(\omega)F_R^*(\omega+\Omega) - F_R^*(\omega)F_R(\omega-\Omega)\Big]. \tag{3.12}$$

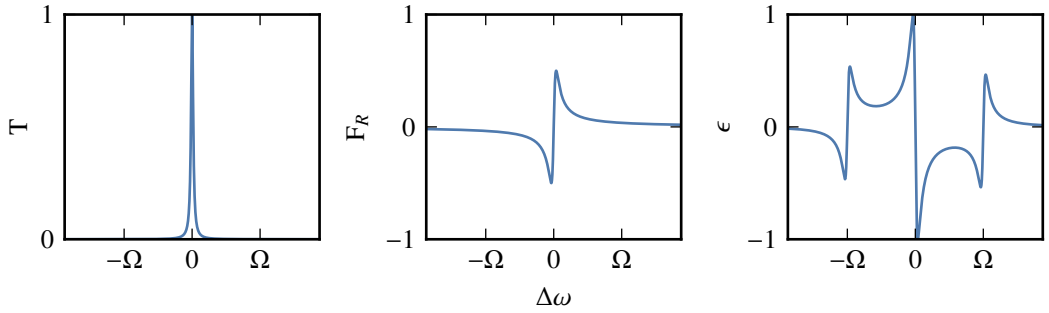


Figure 3.5: Simulated plots of cavity transmission function (Equation 3.4), reflection coefficient (Equation 3.7) and PDH error signal (Equation 3.12) from left to right, as a function of $\Delta\omega$, the angular frequency difference between the cavity resonance and the laser frequency. Ω is the modulation frequency.

An example PDH error signal is shown in Figure 3.5. The steep antisymmetric slope about the resonance is ideal for frequency stabilisation and the large region, equal to twice the modulation frequency, about the resonance for which the signal is of the correct sign allows for a large capture range.

PDH locking is generally achieved with the modulation applied to the laser beam using an EOM with the beam then incident on a cavity. The light reflected from the cavity is then measured by a photodetector and the signal from the detector is mixed with the same frequency source as is driving the EOM. A low-pass filter removes the higher frequency component from the resulting signal which is passed to the servo system which attempts to stabilise the laser frequency. This is shown in Figure 3.4.

3.4 Polarisation Spectroscopy

PDH is a powerful technique but is not atomically referenced and is fairly complicated to implement, particularly if it is to be used to its full potential. Polarisation spectroscopy (PS) is an alternative frequency stabilisation technique that is atomically referenced and able to provide impressive linewidth reduction, which will be demonstrated here. Discussions of the advantages and performance of PS are given in the following sections.

Shown in Figure 3.6 is a schematic of an implementation of PS. In PS a circularly polarised pump beam from a monochromatic laser, with frequency close to an atomic resonance, induces frequency-dependent circular birefringence in a magnetically-shielded atomic gas sample. A linearly polarised beam from the same laser source is used to measure the birefringence, monitored with a balanced polarimeter consisting of a half-wave phase retarder, PBS and two detectors. The magnetic shielding of the atomic gas sample is required to prevent Faraday rotation caused by ambient magnetic fields affecting the probe beam polarisation.

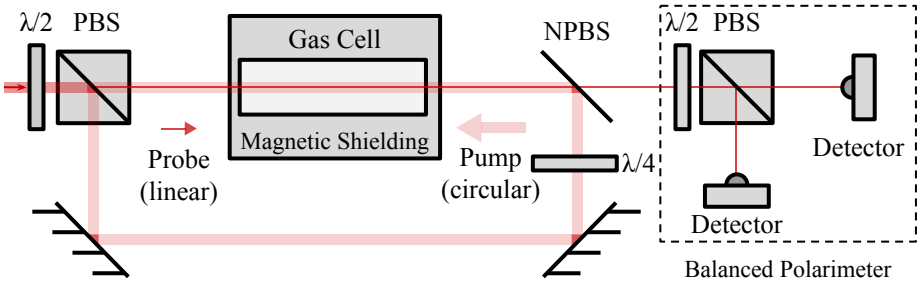


Figure 3.6: A schematic of polarisation spectroscopy with a balanced polarimeter. The power balance between the probe and the pump beam is controlled with the left-most $\lambda/2$ phase retarder and polarising beamsplitter (PBS). The $\lambda/4$ retarder is adjusted to produce a circularly polarised pump beam. The non-polarising beamsplitter (NPBS) is used to counter-propagate the pump beam through the atomic sample without altering the polarisation of the circular pump or linear probe. The final $\lambda/2$ retarder, PBS and the detectors form the balanced polarimeter that monitors the polarisation rotation of the probe.

The circularly polarised pump beam induces circular birefringence in the atomic sample by partial optical pumping of the sample into one of the extreme hyperfine sublevels, $m_F = \pm F$, where m_F labels the hyperfine sublevel and F is the atomic angular momentum number. This partial optical pumping, referred to here as the anisotropy of the medium, results in unequal absorption coefficients for each circular polarisation. The linearly polarised probe beam can be decomposed into two equal and oppositely circularly polarised components which undergo different absorption due to the anisotropy. When the circularly polarised components are recombined, after passing through the atomic sample, the probe beam becomes elliptically polarised and rotated from the original linear polarisation angle.

The dispersive error signal is the difference of the two orthogonal polarisation components [113, 122]

$$P_{PS} = P_x - P_y = -P_0 \cos(2\phi + 2\Phi) \tag{3.13}$$

where $P_{x,y}$ are the power of the horizontal and vertical linearly polarised components of the probe after the sample, P_0 is the power of the probe in the absence of a pump beam, ϕ is the angle of polarisation of the probe in the absence of a pump beam and Φ is the additional polarisation rotation of the probe due to the birefringence induced by the pump. The largest PS spectrum is produced when $\phi = \pi/4$ and since Φ is small Equation 3.13 becomes

$$P_{PS} = 2P_0\Phi. \tag{3.14}$$

The polarisation rotation is given by

$$\Phi = \frac{\pi L \Delta n}{\lambda}, \tag{3.15}$$

where L is the length of the atomic sample, λ is the wavelength of the light, $\Delta n = n_+ - n_-$ and $n_\pm$ are the refractive indices affecting the circularly polarised components of the probe beam.

The refractive index and absorption of the medium are related through the Kramers-Kronig dispersion relation to give [97]

$$\Delta n = \Delta\alpha_0 \frac{2c}{\omega_A \Gamma} \frac{\delta}{1 + 4\left(\frac{\delta}{\Gamma}\right)^2}. \tag{3.16}$$

Here c is the speed of light, $\delta = \omega_L - \omega_A$ is the detuning of the laser from the resonance, $\omega_{L,A}$ are the angular frequency of the laser and the atomic resonance, and Γ is the inverse lifetime of the excited state of the resonant transition. $\Delta\alpha_0$ is the difference in absorption coefficients for the circular polarisation components at zero detuning. $\Delta\alpha_0$ is the sum over all m_F ground states of the difference between absorption coefficients for each circular polarisation, with $\delta = 0$, weighted by the ground and excited state population differences, $\mathcal{P}_{F,m_F} - \mathcal{P}_{F',m_{F\pm1}}$;

$$\begin{aligned} \Delta\alpha_0 \quad = \sum_{m_F=-F}^{+F} \Big[\quad & \alpha_{(F,m_F \to F',m_{F+1})}(\mathcal{P}_{F,m_F} - \mathcal{P}_{F',m_{F+1}}) - \\ & \alpha_{(F,m_F \to F',m_{F-1})}(\mathcal{P}_{F,m_F} - \mathcal{P}_{F',m_{F-1}}) \Big]. \end{aligned} \tag{3.17}$$

The absorption coefficient for a given transition is $\alpha_{(F,m_F \to F',m_{F\pm1})} = N\sigma(\omega_L)$ where N is the number density of interacting atoms, and $\sigma(\omega_L)$ is the absorption cross section for the transition.

The atomic substate populations established by interaction with the pump beam can be calculated using optical Bloch equations [122]. The PS signal is proportional to the refractive index difference given by Equation 3.16, which describes a steep, background-free antisymmetric dispersive function.

PS produces a dispersion shaped spectrum about the atomic resonance with zero background, as shown in Figure 3.7, and is ideal for laser locking due to the large capture range and steep gradient near the resonance. Unlike PDH, PS acts as an absolute reference as it tied to the frequency of the atomic-ensemble reference.

Balanced Polarimeter

When Wieman and Hänch [51] originally proposed PS polarisation rotation was monitored with a nearly crossed linear polariser as shown in Figure 3.8. The polariser was crossed such that only a small proportion of the probe beam passed through to the detector in the absence of the pump. With the pump inducing anisotropy in the atomic sample the rotation of the probe could be detected after the polariser.

Pearson *et al.* proposed the alternative method of using a balanced polarimeter, shown in Figure 3.6, which provides a background-free signal with peak-to-peak height more than an order of magnitude greater than with the crossed polariser method [99, 113].

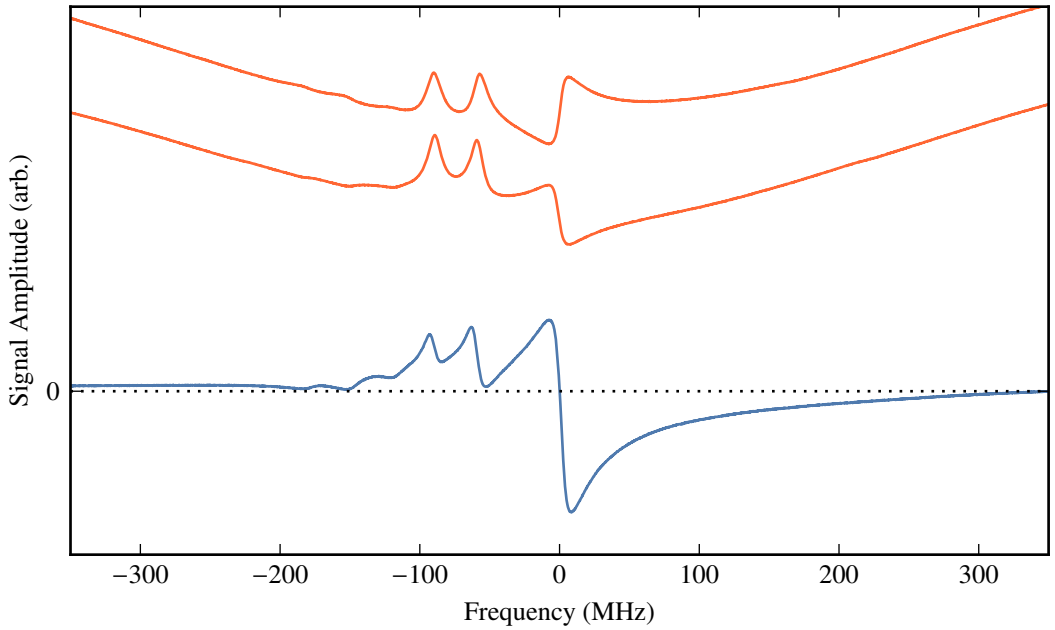


Figure 3.7: Measured absorption spectrum for PS horizontal and vertical polarisations (red) and the corresponding error signal (blue). This spectrum is for the rubidium-85 D2 line with the horizontal axis zero relative to the $F = 3$ to $F' = 4$ transition.

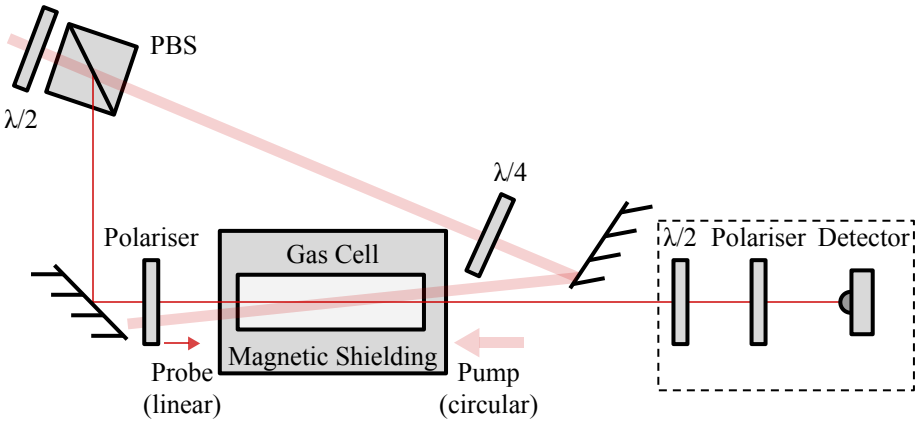


Figure 3.8: A schematic of the PS setup initially suggested by Wieman and Hänsch [51]. The polariser by the detector is a linear polariser crossed with the linear polariser before the gas cell.

Concentric Beams

Wieman and Hänch's initial proposal for PS used a mirror next to the probe beamline to angle the pump beam such that it was *almost* concentric with the probe beam, as shown in Figure 3.8. In saturated absorption spectroscopy imperfect alignment of the pump and probe results in geometric broadening of the spectral features and a reduction in the overlap region and thus the number of atoms involved and the strength of the signal [123] and PS is similarly affected.

Many saturated absorption spectroscopy setups use arrangements similar to the PS setup shown in Figure 3.8 in order to get the pump and probe beam to be approximately concentric and counter-propagating however the layout shown in Figure 3.2 provides a much simpler arrangement with concentric and counter-propagating beams and fewer required optics. Similarly the PS layout has been improved with the use of a non-polarising beam splitter to reflect the pump beam into the gas cell without interfering with the pump's circular polarisation or rotated linear polarisation of the probe. Concentric beams give the pump and probe the maximum interaction volume, minimises geometric broadening, and maximises the signal produced.

3.4.1 Error Signal Characteristics

There are a number of aspects of PS that make it an attractive candidate for high-bandwidth frequency stabilisation.

High Signal-to-Noise Ratio

PS has a number of advantages with respect to SNR compared to other techniques. PS is a velocity selective technique that produces a dispersive error signal with high frequency-discrimination slope, comparable to SA and MTS. The subtraction associated with the balanced polarimeter removes technical noise, particularly probe laser intensity noise, across the entire signal bandwidth. The resultant PS error signal therefore approaches the shot-noise limit and the SNR is very high over a large bandwidth, shown in Figure 3.18.

Modulation Free

PS is a technique that does not require any modulation of the laser beam or electronic signals. With modulation-based frequency stabilisation techniques such as PDH, SA, or MTS, the feedback bandwidth is theoretically limited to half the modulation frequency by the Nyquist theorem. In practise the bandwidth is limited to well below the Nyquist limit due to the need to filter the signal to remove the second harmonic product and upconverted flicker noise at the first harmonic. The filtering is made more difficult in servo applications by the need to keep latency low to preserve phase margin. The absence of modulation also reduces the implementation complexity and cost but some care is required to minimise noise contributions at low frequencies which cause frequency drift.

Selectivity

PS optically pumps the atomic sample into the primary transition pair, for example the $5^2S_{1/2}$ ($F = 3$) $\rightarrow$ $5^2P_{3/2}$ ($F' = 4$) transition of Rb85. Subsidiary transitions, such as the $F = 3$ $\rightarrow$ $F' = 3$, are strongly suppressed and the crossover resonances that dominate SA become comparatively small and do not cross zero. This selectivity, which also applies to MTS, has two important benefits. First the PS error signal is of the correct sign over the full Doppler with of the sample, free of the zero crossings at nearby resonances that are seen with other techniques such as SA, allowing for high effective feedback bandwidth.

Second, if a laser locked to a PS feature becomes unlocked, for example due to an external disturbance, there are no zero-crossings in the feedback error signal to trap the locking servo anywhere but at the desired primary transition. This defines the capture range; that is, the range of frequencies about the resonance for which the error signal is of the correct sign to provide negative feedback. The capture range can be deduced from the error signal by locating the zero-crossings to either side of the resonance as shown in Figure 3.9. For the spectra shown in Figure 3.9 the PS capture range is greater than ±150 MHz, many times larger than that the ±15 MHz of SA.

3.5 Polarisation Spectroscopy Performance

Visible light has frequencies in the THz range which is well beyond the capabilities of modern electronics to directly measure so a number of clever strategies have been developed to measure the spectral quality of laser systems. The performance of PS frequency stabilisation with high-bandwidth feedback has been characterised using a number of methods detailed here. Spectral linewidth is a useful metric by which stabilisation techniques are compared and it can be considered over a range of timescales with different implications. Short timescales measurements, less than a second, indicate the narrowness of the frequency spectrum and are useful when considering frequency dependent interactions, such as those with atoms or optical cavities. Long timescale measurements, minutes to hours, are useful to determine the stability of the laser and its susceptibility to frequency drift and the robustness of the locking scheme.

3.5.1 Laser Systems

The basic layout of the laser system is shown in Figure 3.10. A number of the measurements described in this chapter required only a single laser however the most reliable method, two-laser heterodyne (see Section 3.5.2), requires two very similar laser systems. Two almost identical laser systems were constructed for these measurements with the only major difference being the ECDL itself, one being a MOGLabs ECD003 and the other a Toptica DL pro. Both lasers

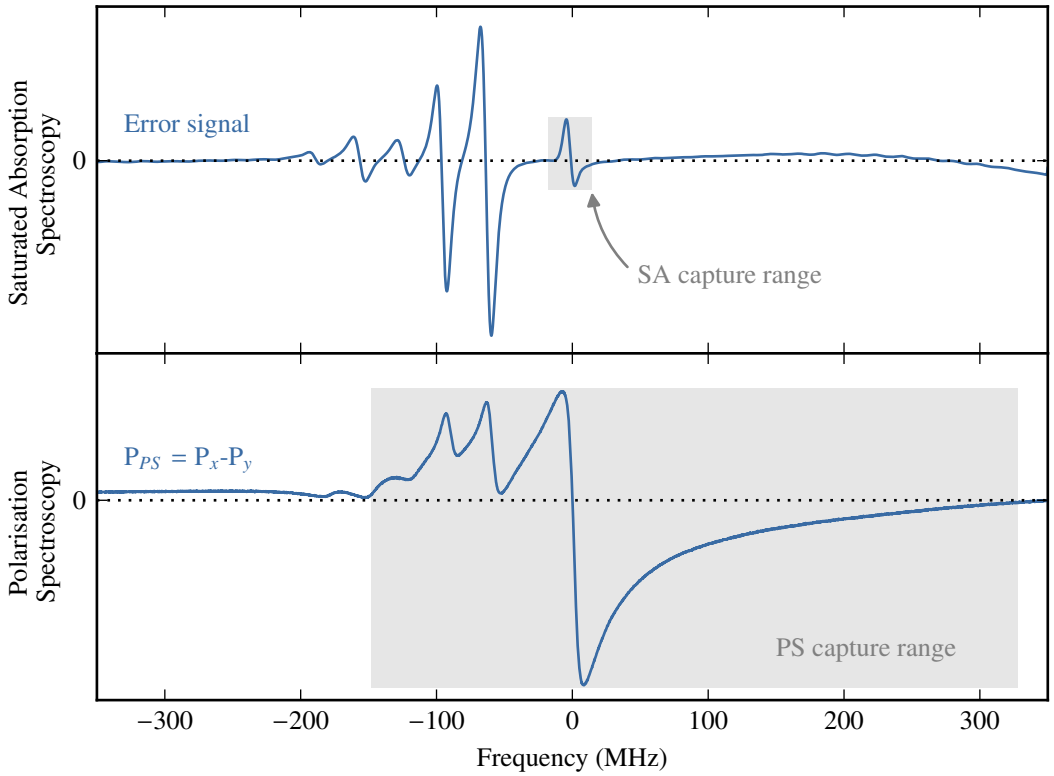


Figure 3.9: Error signals generated by saturated absorption spectroscopy (top) and polarisation spectroscopy (bottom) for the Rb85 D2 transition. The shaded regions indicate the capture range of the respective error signals, the SA capture range is ±15 MHz and the PS range is greater than ±150 MHz.

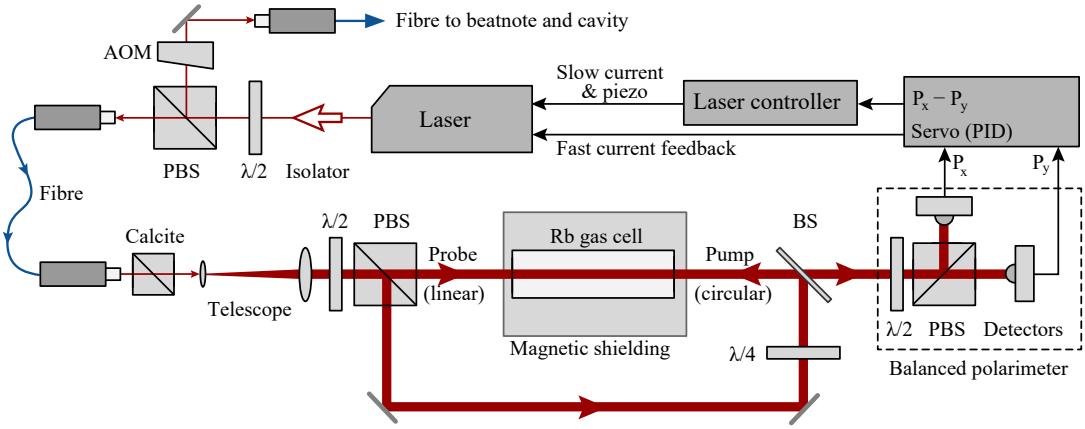


Figure 3.10: Schematic of the polarisation spectroscopy apparatus used to measure the performance of the high-bandwidth feedback. The beam from the laser passes through an isolator before being split into two beams by a PBS and coupled into optical fibres. One fibre leads to the PS setup shown here, the other to the beatnote or cavity measurements via an AOM. The PS setup consists of a polarisation stabilising Glan-Thompson prism followed by a beam expanding telescope. The expanded beam is then divided by a PBS into a linearly polarised probe and circularly polarised pump which counter-propagate, via a non-polarising 50:50 beam splitter (BS), through the magnetically-shielded 15 cm-long rubidium gas sample. The polarisation rotation of the probe beam is then measured by a balanced polarimeter which consists of a $\lambda/2$ waveplate, PBS and two high-bandwidth photodetectors (Thorlabs PDA10A).

were controlled by MOGLabs laser controllers[1] and used high-bandwidth, 14 MHz, servo controllers[2] to control the high-frequency diode injection current channel (>10 MHz bandwidth). The laser controllers were used as servos to provide the low-frequency feedback to the ECDL piezo (1 kHz bandwidth) and slow current channel (50 kHz bandwidth). To provide flexibility when performing the numerous measurements with different locking techniques the beam from each laser was split in two and coupled into single-mode polarisation-maintaining fibres. For each laser, one beam also passed through a double-pass AOM before being coupled into the fibre to provide some frequency control which was useful for both cavity and heterodyne measurements.

In the two near-identical polarisation spectroscopy setups, high quality calcite polarisers where used to stabilise the polarisation of the beam out of the fibre because ambient temperature changes resulted in polarisation changes in the output beam from the fibres, even though polarisation maintaining fibres were used. Following the calcite a telescope expanded the beams to the size of the apertures in the double-layer mu-metal magnetic shields, 1.5 cm diameter. Expanding the beams reduced transit-broadening and improved the optical pumping

[1] MOGLabs laser controlled with a MOGLabs DLC202 and Toptica laser with a DLC252.
[2] NewFocus LB1005, 14 MHz bandwidth

within the 7.5 cm long rubidium gas cells. The pump and probe beams had typical powers of 2.4 mW and 2.7 mW respectively. High-bandwidth photodetectors[3] where used in the balanced polarimeter to generate the error signal.

3.5.2 Heterodyne Methods

Heterodyning is a technique, invented by Reginald Fessenden in 1901, which mixes two frequencies to produce a new frequency [124]. The technique can be used to examine the spectral properties of lasers as the newly produced frequencies can be tailored such that they are easily measurable by photodetectors and spectrum analysers.

Laser frequency spectrum measurements using heterodyne technique are simple to implement with the correct equipment. The two laser beams are combined such that they co-propagate, requiring some mirrors and a beamsplitter/combiner, and then directed onto a high-bandwidth photodetector. The signal from the detector can then be fed into a spectrum analyser which will display the power spectral density of the light on the detector.

Heterodyne measurements can also be performed with sound waves, in fact this method is used to tune musical instruments as the new frequency produced can be heard as a 'beat' or 'beatnote'. The term 'beatnote' is also used in general to refer to the signal resulting from a heterodyne measurement.

Basic Theory

In the electrical signal context heterodyning involves the 'mixing' or multiplying of two signals (e.g. two sine waves) to produce two different signals with frequencies equal to the difference and sum of the original frequencies:

$$\sin(\theta_1)\sin(\theta_2) = \frac{1}{2}\cos(\theta_1 - \theta_2) - \frac{1}{2}\cos(\theta_1 + \theta_2). \tag{3.18}$$

In the optical context this is achieved due to the interference term accrued when squaring the electric field in order to calculate the intensity detected by the photodetector. The intensity of an electric field is given by:

$$I(t) = \frac{c\epsilon_0}{2}|E|^2. \tag{3.19}$$

For two co-propagating lasers with electric fields $E_{1,2}$ and angular frequencies $\omega_{1,2}$ we can write

$$E_i(t) = \sin(\omega_i t). \tag{3.20}$$

The electric field at the detector, E_T, is given by the sum of E_1 and E_2, and thus the intensity is given by,

$$\begin{aligned} I(t) &= |E_1(t) + E_2(t)|^2 \\ &= |E_1(t)|^2 + E_1(t)E_2^*(t) + |E_2(t)|^2. \end{aligned} \tag{3.21}$$

[3] Thorlabs PDA36A-EC, 150 MHz bandwidth

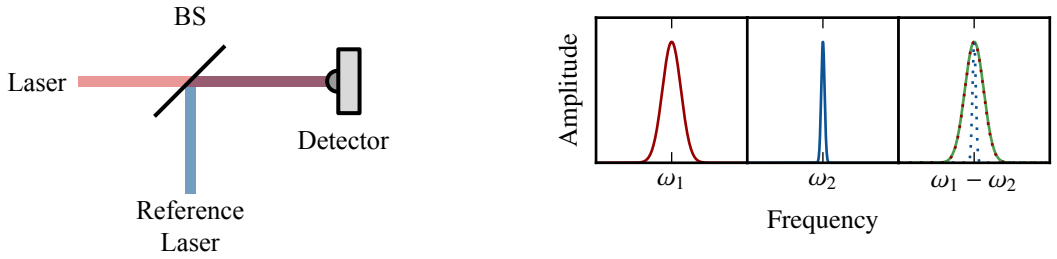


Figure 3.11: Heterodyne measurement with a spectrally narrow reference laser by combining the laser of interest (ω_1, red) with the reference laser (ω_2, blue). Shown on the right is the spectral lineshapes of the lasers and the heterodyne beatnote (green).

The interference term, $E_1(t)E_2(t)$, allows for the spectral measurements with optical signals as

$$\begin{aligned} E_1(t)E_2(t) &= \sin(\omega_1 t)\sin(\omega_2 t) \\ &= \frac{1}{2}(\cos([\omega_1 - \omega_2]t) - \cos([\omega_1 + \omega_2]t)), \end{aligned} \tag{3.22}$$

and if $\omega_{1,2}$ are appropriately selected then the first term in Equation 3.22 is measurable. The difference in frequency ideally should be fairly small, less than a few GHz, which can often be induced on lasers with the same frequency by an AOM or EOM.

The equations presented above provide an obviously naïve approach as the frequency spread of the lasers is non-zero but gives a sufficient demonstration of the basis of the heterodyne method. The spectral profile of the heterodyne signal is formed by the convolution of the spectral profiles of the component lasers, thus strategies are required in order to resolve the lineshape of a single laser. For two near identical lasers with Gaussian lineshape the beatnote FWHM is $\sqrt{2}$ larger than the individual laser FWHM linewidth and $\sqrt{2} \times 2\sqrt{2\log_e 2} = \sqrt{2} \times 2.35$ larger than the individual laser RMS linewidth.

Frequency Reference

One method of resolving the lineshape of a single laser is to use a second laser known to have a frequency lineshape much narrower than the laser of interest so that the convolved lineshape is the same as the lineshape of the laser of interest. This method is shown in Figure 3.11.

The initial intention with the experiments described in this chapter was to be able to use a PDH locked laser as a frequency reference to characterise the performance of another laser locked with PS using high-bandwidth feedback. Unfortunately, initial measurements were unable to confirm that the PDH locked laser was in fact much narrower than PS locked laser so more sophisticated methods were required.

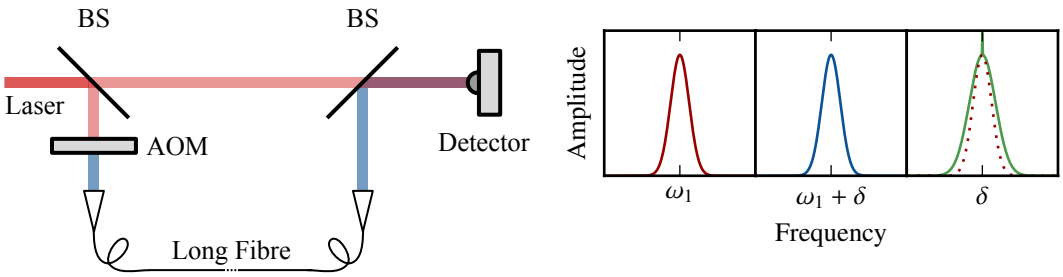


Figure 3.12: Self-heterodyne measurement. The laser is split with one beam frequency shifted by an AOM from the initial frequency (ω_1, red) to an offset frequency ($\omega_1 + \delta$, blue) to enable measurement by a spectrum analyser and then propagated through a long optical fibre to such that the beams are no longer correlated. Shown on the right are the spectral lineshapes of the laser and the heterodyne beatnote (green).

Self-heterodyne

The self-heterodyne technique involves beating a laser with itself in order to perform a heterodyne measurement, as shown in Figure 3.12 [125]. In order to perform this measurement it is necessary to split the laser into two beams and frequency shift one beam, usually with an AOM, such that the beatnote is not centred at DC. The two beams are still correlated so it is necessary to propagate one beam to reduce the coherence otherwise the lineshape of the beatnote would just appear as a delta function centred at the AOM frequency [126]. The beam is typically delayed by a very long optical fibre ideally such that the beam is delayed for as long as the coherence time of the laser. Practically this would require optical fibres with lengths from 1 to 100 km depending on the linewidth of the laser under investigation. For example to have minimal coherence between frequency components at 1 kHz would require a fibre with a length of 300 km. Unfortunately the absorption losses in fibres tends to be at least a few dB per kilometre making fibres longer than 10 km troublesome. Shorter fibres can be used with more complex analysis, as shown by Reference [126].

Self-heterodyne was used to measure the performance of PS. Figure 3.13 shows measurements of a self-heterodyne beatnote from a laser locked with PS without high-bandwidth feedback. The figure shows the beatnote at various ranges and resolutions with the full beatnote, the central portion of the beatnote, and a close up of the central peak. The centre of the beatnote consists of a central portion with a frequency FWHM of 33 kHz, an extremely narrow peak with a FWHM of 240 Hz and an even narrower small peak with a width under 100 Hz. The central portion has a width comparable to the PS linewidth reported in Reference [98], 20 kHz, and it was unclear which of the peaks might correspond to the laser frequency spread, residual correlation between the two beams or laser intensity modulation from the AOM. Due to this uncertainty it was necessary to perform additional measurements to determine the true

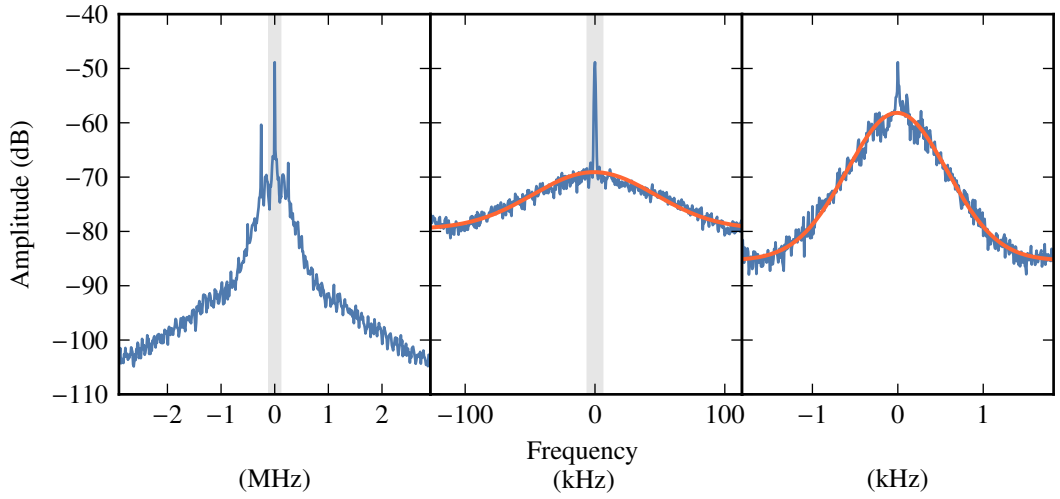


Figure 3.13: An example of a self-heterodyne beatnote of an ECDL locked with PS at various resolutions. The grey shaded areas indicate the frequency range of the next figure as the resolution is increased (resolution bandwidth is, left-to-right, 3 kHz, 1 kHz, 1 Hz). The red lines indicate Gaussian fits to the beatnote features, the middle fit has a -3 dB width (FWHM) of 77 kHz and the right fit a width of 570 Hz which correspond to a laser FWHM linewidth of 33 kHz and 240 Hz respectively. The horizontal axis indicates the frequency from the beatnote centre.

linewidth achievable with PS.

Two-Laser Heterodyne

Two almost identical laser setups can be used in a heterodyne measurement to determine the lineshape under the assumption that the lineshape of each laser is the same. This method is very similar to the self-heterodyne technique without the need to consider the coherence between the two beams. As with self-heterodyne it is necessary to frequency-shift one of the beams with an AOM such that the beatnote is not centred at DC. A summary of two-laser heterodyne is shown in Figure 3.14.

Two-laser heterodyne provides a useful measurement due to its simplicity and the lack of confounding effects (such as those found with self-heterodyne) and thus can provide greater confidence in the results of the measurement. The downside of this method is the obvious requirement for two very similar lasers frequency stabilised with the same methods.

In order to test the efficacy of high-bandwidth PS two similar laser setups were used; both used commercial Littrow configuration lasers, identical optic setups as shown in Figure 3.10 with optical fibres that can be connected to the diagnostic measurements such as the heterodyne setup and the high-finesse cavity.

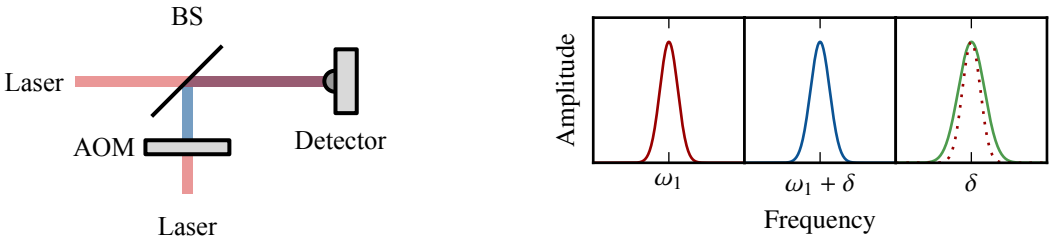


Figure 3.14: Heterodyne measurement with two 'identical' lasers. One beam is frequency shifted by an AOM from the initial frequency (ω_1, red) to an offset frequency ($\omega_1 + \delta$, blue) to enable measurement by a spectrum analyser. Shown on the right is the spectral lineshapes of the lasers and the heterodyne beatnote (green).

One laser was frequency shifted with a double-pass AOM, to give a total frequency shift of 160 MHz, which was then combined with the other laser with a 50:50 beamsplitter and then incident on the photodetector. The 1 GHz detector[4] was connected to a spectrum analyser[5] to measure the spectrum of the beatnote that was centred around 160 MHz due to the frequency shift imposed on one beam of the heterodyne measurement by the AOM. An example of a beatnote from two lasers locked with PS with high-bandwidth feedback is shown in Figure 3.15. The majority of the optical power is contained within the central peak of the beatnote which, unlike self-heterodyne, can confidently be said to be due only to the frequency spectrum of the lasers. The central peak has a −3 dB width (FWHM) of 2.0±0.4 kHz. As the lasers are uncorrelated and assuming they have identical Gaussian lineshapes indicates that the individual lasers have an RMS linewidth of 0.6±0.1 kHz. The 'shoulders' of the beatnote at ±1.5 MHz correspond to the servo bump of the fully locked noise spectrum shown in Figure 3.18. These measurements indicate that PS with high-bandwidth feedback is able to narrow the laser linewidth to 0.6±0.1 kHz which is a significant improvement, by two orders of magnitude, over previously published PS performance [98].

The measurement shown in Figure 3.15 is a 50-shot average with a total measurement time of approximately 2 seconds. Individual measurements, with measurement times around 40 ms, had -3 dB widths as low as 880±280 Hz, corresponding to an RMS laser linewidth of 270±90 Hz.

These measurements show, for the first time, PS achieving sub-kilohertz linewidth narrowing, a regime previously relegated to the more complex PDH. PS is able to achieve this low linewidth without the need for modulation or a high-finesse cavity while providing an absolute reference.

[4]NewFocus 1621

[5]9 kHz to 7 GHz Rohde and Schwarz FSP7

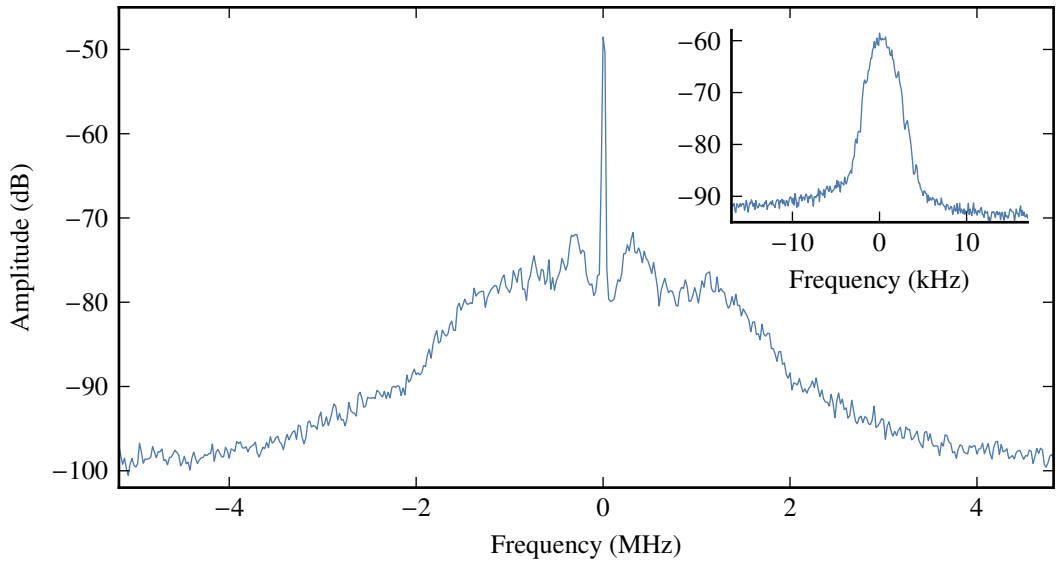


Figure 3.15: Heterodyne beatnote for two lasers locked with PS and inset is a higher resolution measurement of the centre peak which has a −3 dB width (FWHM) of 2.0±0.36 kHz which corresponds to a laser RMS width of 0.6±0.1 kHz. Both figures are 50-shot averages captured with resolution bandwidths of 30 kHz and 100 Hz and total measurement times of 0.5 s and 2 s respectively.

3.5.3 Cavity Diagnostics

To confirm the linewidth narrowing results shown by the heterodyne method, we used a high-finesse optical cavity to directly measure the frequency spectrum of a single laser. Although FM demodulation of the heterodyne beatnote can be used to determine the frequency noise spectrum [127], wide bandwidth FM demodulation is technically difficult and the results are still a convolution of the noise spectra of two lasers. A cavity readily provides the noise spectrum for a single laser. The high-finesse optical cavity used for PDH frequency stabilisation provides a useful diagnostic tool for examining laser spectral behaviour. There are a large number of ways that the cavity can be used to examine the spectral behaviour of a laser system; two that were particularly useful are detailed in this section.

Cavity Resonance Sweep

The high-finesse optical cavity provides a useful tool for quickly checking the efficacy of high-bandwidth locking if the cavity linewidth and final laser linewidth are within a few orders of magnitude of each other. While attempting to lock a laser, the transmission of a portion of the laser beam through the cavity can be monitored. By sweeping the diagnostic beam frequency incident on the cavity across the cavity resonance, with an AOM, the laser frequency jitter can be monitored as the bandwidth of the locking scheme is increased. An example of

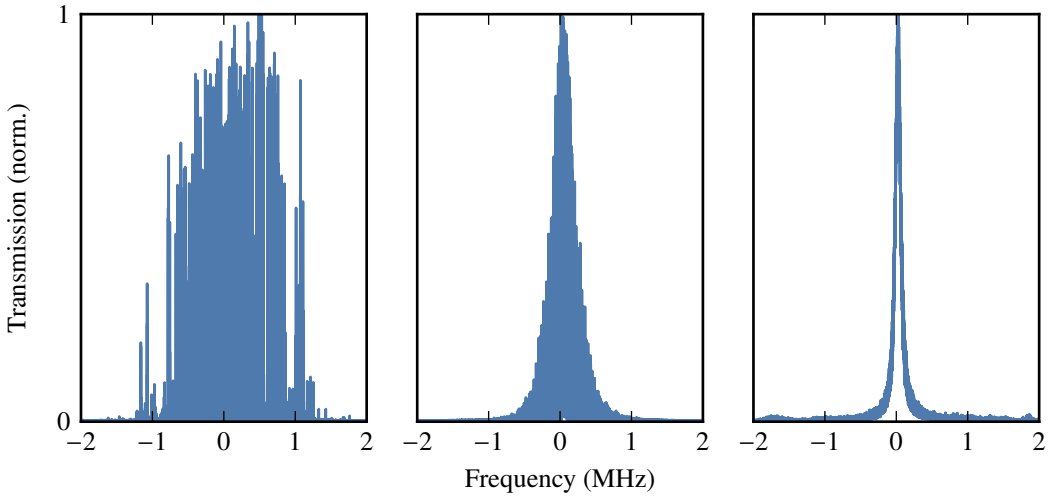


Figure 3.16: Optical cavity transmission as a function of frequency offset from the cavity resonance for varying laser locking bandwidths. The frequency offset was scanned using a double-pass AOM. Piezo only laser feedback (left), piezo and slow current feedback (middle), and piezo, slow current and fast current feedback. Cavity FWHM 71.6 kHz.

this process is shown in Figure 3.16 where the laser is locked with PS and the bandwidth of the servo system is varied. The resulting traces provide a clear illustration of the effect of increased locking bandwidth. With piezo-only locking the peak appears broad as the laser jitters about the resonant frequency; with slow-current feedback the transmission peak shape becomes apparent. Finally, with high-bandwidth feedback the transmission function of the cavity is clear with the peak width limited by the cavity finesse indicating that the laser linewidth is much smaller than the cavity linewidth.

One-Sided Peak Measurements

If the linewidth of the laser is lower than that of the optical cavity then the cavity can be used as a frequency discriminator, to convert frequency noise to electrical noise which can then be analysed to determine the laser frequency noise spectrum and linewidth. As shown in Section 3.3.1, the transmission through the cavity, as a function of laser frequency, can be described by an Airy function,

$$T(f) = \frac{1}{1 + F\left(\frac{\omega L}{c}\right)^2}. \tag{3.23}$$

For a relatively narrow linewidth laser, the frequency of the laser into the cavity can be adjusted (with an AOM or frequency sidebands with an EOM) such that the transmission through the cavity is approximately 0.5, thus any deviations in frequency will manifest as a change in the transmission through the cavity. The variation of the signal measured by the detector can be

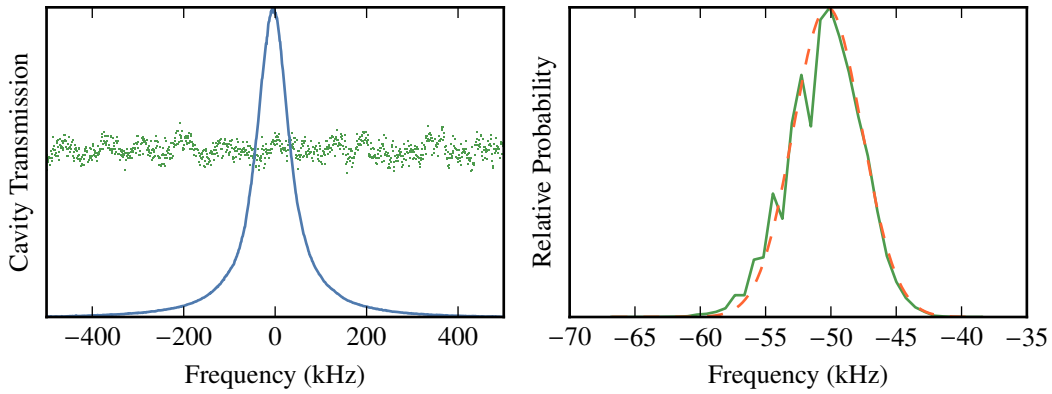


Figure 3.17: Laser linewidth measured by measuring the transmission through the cavity. On the left figure the blue line shows the cavity transmission as the frequency of the beam into the cavity is scanned with an AOM and the green points show the cavity transmission when the AOM is used to centre the laser frequency part way up the transmission peak. The green points are from a time series independent of the x axis of that figure. The known transmission function (blue line) can be used to map the transmission distribution to a frequency distribution, shown as the green line in the right figure. The dashed red line is a Gaussian fit to the frequency distribution with an RMS width of 2.4±0.3 kHz. The laser used with this measurement was locked with PS and high-bandwidth feedback.

mapped to the associated variation in frequency using the above equation which can then be used to determine the frequency lineshape as shown in Figure 3.17.

The cavity transmission actually includes contributions from both the frequency and amplitude noise of the laser. An estimate of the linewidth contribution from amplitude noise requires an almost identical measurement without the frequency discrimination provided by the cavity. By placing the detector in front of the cavity and ensuring the power incident on the cavity is the same as the average power of the cavity transmission measurement, the amplitude noise contribution to the linewidth can be determined. This measurement indicates that the amplitude noise contribution to the RMS linewidth was 1.4±0.2 kHz. The analysis assumes that the frequency and amplitude noise are uncorrelated and that the observed signal is a convolution of the two. The linewidth determined from the cavity transmission mapping therefore consists of the 1.4±0.2 kHz linewidth amplitude noise convolved with the frequency noise linewidth of 2.0±0.5 kHz to produce the frequency distribution shown in Figure 3.17, which has an RMS width of 2.4±0.3 kHz.

3.5.4 Frequency Noise Measurements

Noise measurements can be useful when attempting to identify frequency-specific noise sources and for determining the bandwidth of frequency locking schemes. Frequency spectra of the

laser can be acquired from the on-resonance PS error signal or the cavity transmission signal part way up the Airy peak (similar to that used in Section 3.5.3). The frequency spectrum of the PS error signal provides a measure of the laser frequency noise in combination with the PS frequency discrimination, photodetector, and electronic noise and gain. The measurements shown in this section were generated by stitching frequency spectra measured with a high-bandwidth radio frequency (RF) spectrum analyser[6] and a computer-controlled high-dynamic-range audio digitiser[7].

Figure 3.18 shows a number of frequency noise measurements captured from a laser locked with PS. The RF spectrum analyser was calibrated using the slope of the error signal on-resonance, which itself can be calibrated by identifying features in the spectrum such as the 31.5 MHz difference between the two rubidium-85 crossovers (see Figure 3.2), and the known cavity transmission function as shown in Figure 3.17. The low frequency data were calibrated by matching to the RF spectrum data at 10 kHz. The upper portion of the figure shows the frequency spectrum of the PS error signal for a range of locking bandwidths from an unlocked laser to one fully locked with high-bandwidth feedback. Comparing the unlocked spectrum to that of the other spectra allows for an approximation of the bandwidth of that feedback mechanism by finding the intersection with the unlocked spectrum:

- Piezo-only feedback has a frequency bandwidth of approximately 1 kHz.
- Piezo and slow-current feedback has a bandwidth of approximately 50 kHz.
- Piezo, slow-current and fast-current feedback has a bandwidth of approximately 500 kHz.

The noise floor of this measurement can be measured by examining the spectrum of the PS error signal while the laser is far from the atomic resonance. The spectrum of the fully locked spectrum is coincident with the noise floor from 450 Hz to 350 kHz indicating that the servo system is utilising all the signal available in that region. The frequency of the servo bump at 1.3 MHz for the fully-locked spectrum is consistent with phase lag in the laser diode response to injection current modulation [117].

The high-finesse optical cavity provides another method for examining the frequency noise spectrum of the laser, with a much lower noise floor as shown in the lower plot of Figure 3.18. The cavity was used as an independent laser frequency discriminator by shifting the laser frequency with an AOM such that the transmission through the cavity was half of the peak (similar to the method described in Section 3.5.3) and the spectrum of the transmitted signal was calibrated given the known cavity transmission function. The noise floor for this measurement was determined by illuminating the photodetector directly with light of intensity equal to that used for the cavity transmission measurement but without the frequency dependent influence of the

[6] 9 kHz to 7 GHz Rohde and Schwartz FSP7

[7] 24 Hz to 100 kHz E-Mu E-DSP

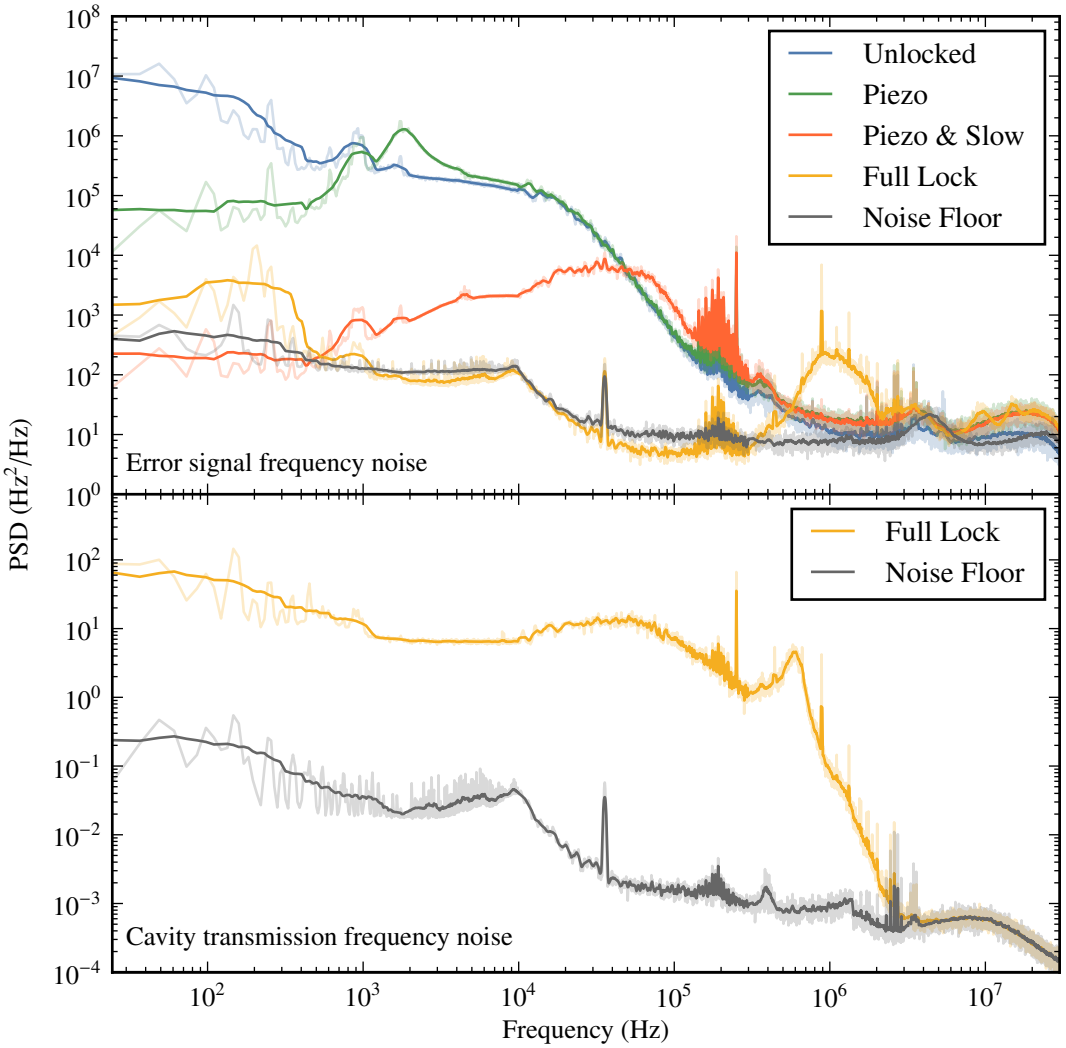


Figure 3.18: Power spectral density (PSD) measurements of a PS locked laser. The top figure is from the PS error signal and the bottom figure is from the cavity transmission signal with laser tuned to half the peak height. The PSD for a variety of scenarios is shown: the unlocked laser with no frequency stabilisation, piezo only feedback, piezo and slow current feedback, fully locked with piezo, slow current and fast current feedback, and the noise floor of the measurement where the laser frequency is far from the atomic and cavity resonances. The measurements are shown with a superimposed smoothed curve with a moving average and window size of $10\log_{10}(f)$ where f is the frequency.

cavity. The signal was well above the noise floor for frequencies up to the 5.5 MHz bandwidth of the photodetector. The noise floor for the cavity measurement is significantly lower than that of the error signal measurement and shows that there is significant signal available that could be used if the noise in the PS error signal could be lowered. The three to four orders of magnitude difference in noise floors is consistent with the lower shot-noise in the cavity measurement, once the power conversion by the measurement is taken into account. The laser power incident on the cavity was 10 μW, compared to the 1 mW in the PS balanced polarimeter. Due to the narrow linewidth of the cavity transmission, 71.6 kHz, it is not possible to measure the frequency noise of the unlocked, piezo-only or piezo and slow current feedback configurations as the laser linewidth was greater than that of the cavity.

Noise Integration

The PSD of a laser, in $\mathrm{Hz}^2/\mathrm{Hz}$, can be described as [127],

$$S(f) = 2\int_0^\infty \langle \Delta\nu(t)\Delta\nu(t+\tau)\rangle \exp(-i2\pi f\tau)d\tau, \tag{3.24}$$

where f is the Fourier frequency, and $\Delta\nu(t)$ is the instantaneous frequency of the laser relative to the mean laser frequency. The RMS linewidth of the laser is then

$$\Delta\nu_{rms}^2 = \int_0^\infty S(f)df. \tag{3.25}$$

Using Equation 3.25 we can extract a laser linewidth from the PSD measurements to provide additional confirmation of the other linewidths measurements. Integrating the PSD of the fully locked laser, measured with the optical cavity, results in an RMS linewidth of 1.68±0.49 kHz.

The noise floor can also be integrated to provide an indication of the noise contribution to the linewidth measurement. For the data shown in Figure 3.18 the noise contributes 0.11±0.05 kHz which is negligible. The measurements shown have a low frequency cutoff of 24 Hz as the spectrum analysers were unable to measure below this point. The discrepancy between this small noise contribution and the 1.4±0.2 kHz amplitude noise contribution in Section 3.5.3 implies that the majority of the amplitude noise occurs below 24 Hz.

3.5.5 Long Term Stability

Polarisation spectroscopy is inherently a DC technique, susceptible to low frequency drift. Drift in the laser power output as the laser alignment drifts, variations in fibre coupling efficiency if fibres are used, variations in the atomic vapour density due to changes in temperature, thermal effects on the waveplates and changes to the electronic gains and offsets can all affect the lock point of PS due to the resulting intensity noise combined with the difficulty in perfectly balancing the polarimeter.

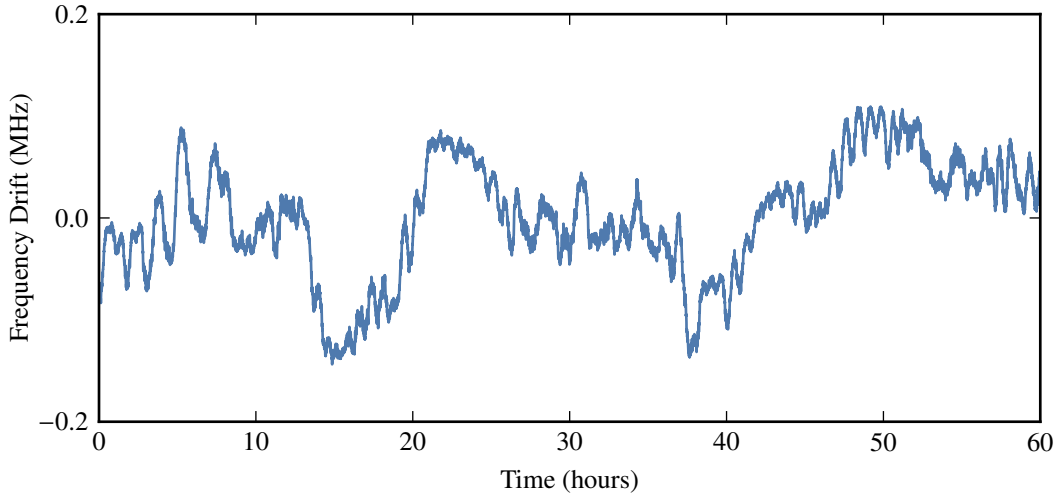


Figure 3.19: Frequency drift of a PS locked laser over a 60 hour period measured every 10 seconds by the high-finesse optical cavity. The standard deviation of this measurement was 51 kHz.

The high-finesse cavity was used as a reference to quantify the long-term drift of the PS locked laser over a period of 60 hours, measured every 10 seconds, see Figure 3.19. Drift in the optical cavity frequency was corrected by reference to the central frequency of a beat-note between a laser locked to the rubidium transition using saturated absorption spectroscopy and the PS locked laser incident on the cavity. The standard deviation of the PS-locked laser frequency measurements was 51 kHz over the 60 hour measurement, approximately half the standard deviation of the measurements in [114] and significantly smaller than the 400 kHz quoted in [128]. The frequency change between each measurement was on average 5 Hz, with a standard deviation of 210 Hz.

The frequency stability of PS is strongly dependent on the extent to which the apparatus is isolated from ambient temperature variations [99]. In our system the lasers are temperature stabilised and isolated with acrylic enclosures, and the optical cavity was temperature controlled and isolated inside a vacuum chamber, but the PS and saturated absorption spectroscopy components and optical components between lasers and optical cavity were not temperature controlled or shielded from the general laboratory environment. The locking stability and drift are expected to improve with environmental isolation of all optical components and temperature stabilisation of the atomic vapour cell. The power into the PS setup is particularly sensitive to polarisation drift in the light exiting the optical fibres. Although single-mode polarisation maintaining fibres were used, they exhibited significant polarisation drift with laboratory temperature variations, and it is expected that shielded free-space propagation or active power stabilisation of the light into the PS setup would further improve the frequency stability.

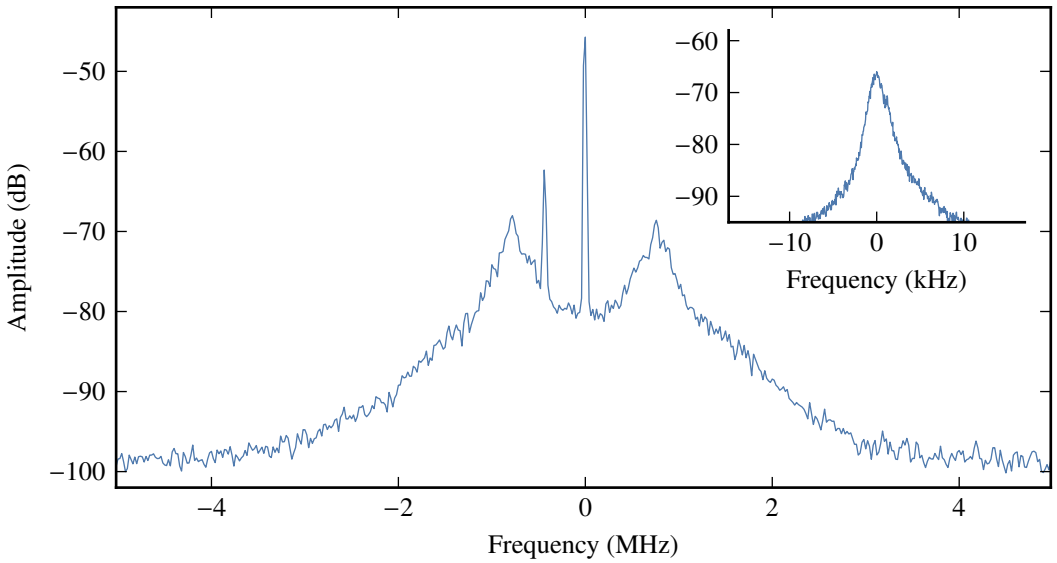


Figure 3.20: Heterodyne beatnote for two cat-eye lasers locked with PS and inset is a higher resolution measurement of the centre peak which has a −3 dB width (FWHM) of 1.2 kHz which corresponds to a laser RMS width of 0.36 kHz. Both figures are 50-shot averages captured with resolution bandwidths of 30 kHz and 100 Hz and total measurement times of 0.5 s and 2 s respectively.

3.6 Cat-eye Laser Beatnote

Sometime after the work described in this chapter was summarised and published in Reference [52] MOGLabs was able to supply two identical new cat-eye configuration lasers[8] that use high-bandwidth low-phase-delay in-laser modulation electronics [129]. These two lasers were inserted into the experimental setup described in Section 3.5.1 in place of the previously used ECDLs. These lasers were able to achieve an individual laser RMS linewidth of 0.36 kHz as shown in Figure 3.20. An interesting feature of this beatnote is the smaller peak to the left of the main peak which is due to residual amplitude modulation caused by the AOM. The modulation from the AOM is usually hidden under the main peak but the lasers used in this measurement had slightly different lock-points and thus had a small frequency offset.

3.7 Conclusion

The various linewidth measurements for PS with high-bandwidth feedback are summarised in Table 3.1. The simplest and most reliable method, two-laser heterodyne, indicates that the laser linewidth achievable with this laser frequency stabilisation technique is 0.6±0.1 kHz, well below previously demonstrated with PS. The difference in linewidth between the cavity one-

[8] MOGLabs CEL

	Method	RMS Linewidth (kHz)
(i)	Cavity one-sided peak	2.0 ± 0.5
(ii)	Cavity transmission integral	1.68 ± 0.49
(iii)	Heterodyne	0.60 ± 0.1
(iv)	Heterodyne (cateye)	0.36
(v)	Long-term drift	51

Table 3.1: Spectral linewidth results from PS locked lasers. (i) Mapping the transmission noise through the optical cavity to the cavity transmission function followed by deconvolving from the amplitude noise. (ii) The results from integrating the power-spectral density of the cavity transmission signal, Figure 3.18. (iii) Laser linewidth derived from the heterodyne measurement, Figure 3.15. (iv) Laser linewidth derived from the heterodyne measurement using cateye lasers, Figure 3.20. (v) Long-term stability measured with the optical cavity, Figure 3.19.

sided-peak and cavity PSD integration measurements can be explained by the low-frequency cutoff of the spectrum analyser used in the integration. The discrepancy between the two-laser heterodyne measurement and the cavity measurements can be attributed to the effects of laser intensity noise on the cavity measurements and the contributions from high-frequency noise which forms the broad 'pedestal' visible in the beatnote but does not affect the heterodyne width.

These measurements indicate that PS is capable of achieving spectral linewidth previously only reachable with expensive high-finesse optical cavities. The long-term frequency stability is easily sufficient for many laser cooling experiments and is significantly lower that previously demonstrated with other laser systems and stabilisation techniques.

Further improvements to spectral linewidth could be made if the noise in the PS measurement can be decreased and the signal strength increased allowing for lower shot-noise and greater noise suppression. Drift can be expected to improve with active temperature stabilisation of the atomic gas cells, active laser-power stabilisation into the PS setup, and free space propagation rather than the use of optical fibres.

The investigations described in this chapter have advanced the understanding of high-bandwidth absolute laser frequency stabilisation which may prove useful in the complex ionisation processes utilised by the CAEIS, laser spectroscopy and laser cooling applications. This research was also instrumental in learning how to utilise high-bandwidth for linewidth narrowing which aided MOGLabs in developing new laser electronics for direct modulation of the diode injection current, and a new fast servo controller[9].

[9]MOGLabs Fast Servo Controller, 40 MHz bandwidth.

Chapter 4

Ultrafast Electron Diffractive Imaging

One of the central motivations for the development of the CAEIS is its potential use as an electron source for the creation of molecular movies [130]. One of the stepping stones towards molecular movies is achieving single-shot ultrafast diffractive imaging. Single-shot electron diffraction from gold nanofoils has already been demonstrated with a cold-atom source [1] but far from the temporal resolution required for UED. With appropriate control of the ionisation pathways, we have previously shown that a CAES can produce ultrafast bunches of cold electrons, as discussed in Section 2.1.4 [1, 46, 71]. Improvements to the bunch current in the next generation of CAES may enable single-shot and ultrafast Bragg diffraction from simple samples, and perhaps coherent diffractive imaging (CDI) [10].

Previous results from the Melbourne CAES have shown diffraction from large crystalline samples using traditional crystallographic techniques [1, 71]. This chapter describes extensions to those results, namely demonstrating ultrafast diffraction from gold nanofoils, and using an additional voltage bias applied to the sample holder to achieve the required beam energy for diffraction from polycrystalline aluminium.

4.1 Crystallography

Crystallography refers to the science of diffractive imaging from crystals and has been studied for over 100 years, being the subject of the 1915 Nobel prize in physics [131]. Crystallographic techniques have been under constant development and refinement since their inception. The majority of crystallography to date has been performed using X-rays however relatively recent developments have utilised electrons. Cryo-electron microscopy has provided another avenue for imaging membrane proteins and other non-crystallisable molecules [132, 133].

Due to the mature understanding of crystallographic techniques, crystallography is ideal for the first steps in demonstrating the capabilities of the CAES. While the CAES is able to operate in CW mode the performance with this apparatus has been optimised for pulse mode (see Section 2.2) and thus the results described in this chapter were taken with pulsed bunches

of electrons.

4.1.1 Theory

The theory of Bragg diffraction is well developed and the basics of the theory are presented here to provide a context for the diffraction results presented later in this chapter [134]. Crystalline structures consist of repeated sub-structures, unit cells, each with identical arrangements of atoms. A perfect infinite crystal can be described as a lattice of unit cells where a set of basis vectors, $\hat{\mathbf{a}}$, $\hat{\mathbf{b}}$, and $\hat{\mathbf{c}}$, can be used to describe translations,

$$\mathbf{t} = u\hat{\mathbf{a}} + v\hat{\mathbf{b}} + w\hat{\mathbf{c}}, \tag{4.1}$$

where the integer coordinates u, v and w describe translation by a number of unit cells along each basis vector from some origin. A primitive unit cell may contain one or more atoms and symmetry is sometimes more apparent if the unit cell consists of multiple primitive unit cells.

When a wave interacting with the crystal lattice interacts with a specific atom in every unit cell then the reflected waves will be in phase which creates a maxima in the diffraction pattern. These planes in the crystal that these atoms line up on can be described by points in the reciprocal lattice. The reciprocal lattice is described by the reciprocal lattice basis vectors, $\hat{\mathbf{a}}^*$, $\hat{\mathbf{b}}^*$, and $\hat{\mathbf{c}}^*$, where

$$\hat{\mathbf{a}}^* = \frac{2\pi\,\hat{\mathbf{b}}\times\hat{\mathbf{c}}}{\hat{\mathbf{a}}\cdot(\hat{\mathbf{b}}\times\hat{\mathbf{c}})}, \qquad \hat{\mathbf{b}}^* = \frac{2\pi\,\hat{\mathbf{c}}\times\hat{\mathbf{a}}}{\hat{\mathbf{a}}\cdot(\hat{\mathbf{b}}\times\hat{\mathbf{c}})}, \qquad \hat{\mathbf{c}}^* = \frac{2\pi\,\hat{\mathbf{a}}\times\hat{\mathbf{b}}}{\hat{\mathbf{a}}\cdot(\hat{\mathbf{b}}\times\hat{\mathbf{c}})}. \tag{4.2}$$

In Equation 4.2, the 2π originates from the convention chosen for the wave-vector, $|\mathbf{k}| = 2\pi/\lambda$. The reciprocal lattice vector is thus

$$\mathbf{g} = h\hat{\mathbf{a}}^* + k\hat{\mathbf{b}}^* + l\hat{\mathbf{c}}^*, \tag{4.3}$$

where the integers h, k, and l are known as the Miller indices.

Scattering events can be described by $\mathbf{q}$, the scattering vector, which is the difference between the initial and final wavevectors $\mathbf{k_0}$ and $\mathbf{k}$;

$$\mathbf{q} = \mathbf{k} - \mathbf{k_0}. \tag{4.4}$$

In the case of elastically scattered waves, $|\mathbf{k}| = |\mathbf{k_0}|$. Constructive interference occurs only where $\mathbf{q}$ is equal to a reciprocal lattice vector, so the scattering condition for crystals is

$$\mathbf{q} = \mathbf{g}. \tag{4.5}$$

The relationship between the scattering vector and the scattering angle is

$$|\mathbf{q}| = 2\,|\mathbf{k_0}|\sin(\theta) \tag{4.6}$$

where θ is the half angle between the initial and final wavevectors. Due to the wavevector convention used the relation between real and reciprocal space distances is

$$|\mathbf{g}| = \frac{2\pi}{d_{hkl}}, \tag{4.7}$$

where d_{hkl} is the distance between the set of planes described by the subscripted Miller indices.

If we combine Equations 4.6 and 4.7 then we get the Bragg condition,

$$2d_{hkl} \sin\theta = n\lambda, \tag{4.8}$$

where n is the diffraction order.

The relative intensity of a wave with scattering vector $\mathbf{q}$ is

$$I(\mathbf{q}) = \left|\tilde{V}(\mathbf{q})\right|^2, \tag{4.9}$$

where $\tilde{V}(\mathbf{q})$ is the Fourier transform of the crystal potential evaluated at $\mathbf{q}$ and we approximate to single-scattering events only.

The Fourier transform of the crystal electronic potential, for an infinite crystal, can be written in terms of the structure factors V_g:

$$\tilde{V}_{inf}(\mathbf{q}) = (2\pi)^3 \sum_g V_g \, \delta(\mathbf{q} - \mathbf{g}), \tag{4.10}$$

where the sum takes the scattering contribution from all the reciprocal lattice points, and the scattering condition is implemented with the Dirac delta. The crystal basis affects the calculation of the structure factors for each reciprocal lattice point. For a particular reciprocal lattice point, the structure factor is calculated by taking the position $\mathbf{x_j}$ and scattering factors $\tilde{V}_j$ for each atom, j, in the crystal basis:

$$V_g = \frac{1}{V_{cell}} \sum \tilde{V}_j(\mathbf{g}) e^{-i\mathbf{g}\cdot\mathbf{x_j}}. \tag{4.11}$$

V_{cell} is the volume of the unit cell. The Fourier transform of an isolated atomic potential gives the scattering factors $\tilde{V}_j(\mathbf{k})$ which are not the same as the Fourier transform of the whole crystal potential, $\tilde{V}(\mathbf{k})$. $\tilde{V}_j(\mathbf{g})$ represents the probability that the atom j will scatter an electron in to a direction corresponding to $\mathbf{g}$. Scattering factors for most elements for a range of scattering angles and electron energies have been tabulated [135]. The numerical values for scattering factors are different for electrons and X-rays and are more commonly referred to as *atomic form factors* in X-ray diffraction.

Due to the relatively low electron energy and bunch charge the CAES it was necessary to use thin samples to achieve detectable transmission of the scattered electrons. The use of thin samples causes a significant deviation from the infinite crystal assumption presented above. For a finite crystal the potential can be calculated by incorporating the *shape function*, $S(\mathbf{x})$, into the calculation:

$$V(\mathbf{x}) = V_{inf}(\mathbf{x}) S(\mathbf{x}) \tag{4.12}$$

where

$$S(\mathbf{x}) = \begin{cases} 1, & \text{for } \mathbf{x} \text{ inside the crystal} \\ 0, & \text{otherwise} \end{cases} \tag{4.13}$$

The delta functions are softened into sinc function to simulate the finite crystal size. The Fourier transform of the finite crystal potential is then

$$\tilde{V}(\mathbf{q}) = \sum_g V_g \, t_x t_y t_z \, \mathrm{sinc}\left(\frac{(q_x - g_x)t_x}{2}\right) \mathrm{sinc}\left(\frac{(q_y - g_y)t_y}{2}\right) \mathrm{sinc}\left(\frac{(q_z - g_z)t_z}{2}\right). \tag{4.14}$$

Here, t_x, t_y and t_z are the sizes of the illuminated portions of the crystal. As we use thin crystal foils in the experiments presented here, the sinc terms end up limiting back towards delta functions in the two large dimensions, x and y. Equation 4.14 indicates that the diffraction condition does not need to be exactly met in order to diffract electrons in a particular direction.

4.1.2 Diffraction Geometry

A typical transmission electron diffraction apparatus uses a collimated beam of electrons directed through the sample and detects the electron flux as a function of diffraction angle. The use of a collimated beam simplifies the treatment of the reciprocal space as any diffracted electrons interacting with a specific reciprocal lattice point will be scattered in the same direction. This results in a series of beamlets which correspond to the reflections where each beamlet is also collimated and of the same size an the incident beam. In order to resolve the beamlets, they can either be propagated to the far field or passed through a lens with the detector at the focal plane.

Propagating the beam to the far field requires satisfying the Fraunhofer condition

$$W^2 \ll \Delta z \, \lambda, \tag{4.15}$$

where W is the transverse size of the initial beam, Δz is the propagation distance and λ is the electron wavelength. The Fraunhofer condition requires either the beam size to be very small or the propagation distance to be very large. Propagation to the far field effectively converts angle to transverse displacement which can also be achieved with a lens and a shorter propagation distance. Lenses are the usual method used in electron microscopy and the lenses are often combined with other sophisticated beam optics allowing for diffraction and imaging of a variety of samples and geometries. The downside is the obvious complexity and cost associated with sophisticated beam optics.

A TEM consist of a relatively simple electron source combined with complex optics and in contrast our CAES consists of a complex electron source with relatively primitive beam optics. For the measurements described in this chapter we only use a simple condenser lens between the source and sample to focus the beam on to the detector (the quadrupole lens discussed in Section 2.4 was added after these measurements were taken). The beam was focused onto

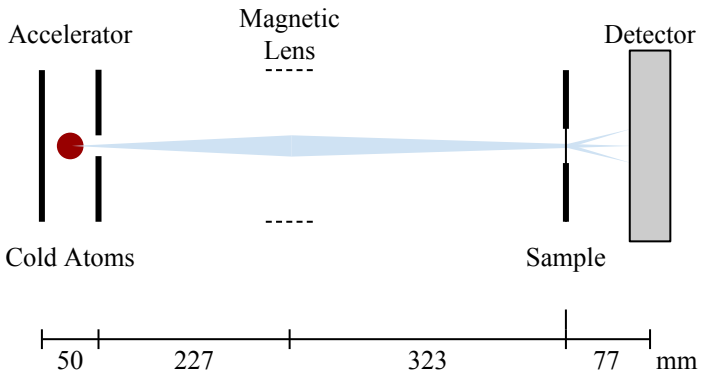


Figure 4.1: A schematic of the apparatus used in the diffraction experiments. The blue area indicates the path of the electrons.

the detector in order to increase the intensity of the Bragg spots to improve the SNR. Due to the non-collimated beam there is a spread of incident angles at the crystalline samples that results in the reflected beamlets possessing the same convergence angle as the incident beam allowing the SNR to be maximised by focusing the beam onto the detector. A schematic of the experimental setup is shown in Figure 4.1.

4.2 Experimental Setup

A typical commercial TEM has an accelerator that operates somewhere in the range between 50–300 kV and is able to image samples with thickness up to around 200 nm. This implementation of the CAES is limited in the energy is can provide to the electrons by electrical breakdown, the substantial engineering required to provide greater beam energies is beyond the scope of the CAES project at this time. Future iterations of the CAES will be able to use the accelerator technology of electron microscopes and particle accelerators to achieve higher beam energy. For the results presented in this chapter the apparatus described in Chapter 2 was used to produce pulsed bunches of electrons. The energy of the beam was 11.7 keV, close to the maximum possible without breakdown.

The majority of the data presented here was generated using electron bunches produced with ionisation pathways designed to maximise beam current. The exception was the ultrafast diffraction measurements which used the ionisation pathways that produced ultrafast electron bunches, as described in Section 2.1.4 [1, 46, 71].

These measurements did not take advantage of the beam shaping potential of the source. Instead, the red excitation laser was adjusted to ensure saturation of the transition across the the atom cloud, to produce a higher beam current than is possible with shaping. The excitation laser beam profile was Gaussian in beamshape with a FWHM of 80 μm at the MOT. Saturation of the excitation resulted in the electron bunches taking on the shape of the atomic cloud, which was

approximately Gaussian. The transverse bunch profile was not of particular concern because it was focused to a point at the detector. Under similar conditions, the shape of the electron bunches was determined to have a Gaussian shape with a FWHM of 1.4 mm with a divergence of $\sigma_{\theta_x} = 0.3\,\mu\text{rad}$ and thus a source emittance of $\epsilon_x = 50\,\text{nm rad}$ [1].

The number of electrons per pulse when producing ns-duration bunches with this setup was measured with a Faraday cup to be 5×10^5 which corresponds to a bunch charge of 80 fC.

The solenoid lens located just after the MOT, well before the samples, was used to focus the electron beam onto the detector with as small a convergence angle as practical. These measurements were taken before the investigations into beam astigmatism discussed in Section 2.4 so there is some asymmetry apparent in the data.

4.2.1 Sample Bias

The maximum electron energy that the accelerator electrodes are able to provide was not sufficient to observe diffraction with some of the test samples. If higher voltages were applied to the electrodes then they would undergo electrical breakdown making it impossible to generate higher energy electrons using the normal accelerator structures. In order to provide a greater range of bunch energies a high voltage feedthrough was attached to the aluminium sample paddle so that incoming electrons would undergo a change in energy, dependent on the polarity and strength of the voltage bias. The sample voltage bias could be up to 8.5 kV before breakdown in the high voltage cable. With 11.7 keV source energy and 8.5 keV on the sample the effective beam energy was 20.2 keV allowing acquisition of diffraction data for aluminium with 31 nm thickness.

4.3 Results

Presented in this section are a number of results demonstrating the capabilities of the CAES with crystalline samples. This work presents the capstone to the diffraction studies conducted with this iteration of the CAES demonstrating diffraction from gold and aluminium samples and ultrafast diffraction.

4.3.1 Transmission Diffraction from Gold

The most successful diffraction was observed with a thin foil sample of single crystal gold. The sample was a standard 3 mm TEM sample of 11 nm thick monocrystalline gold on a carbon grid. Example diffraction patterns from the gold sample are shown in Figure 4.2 which shows 100-shot registered-averages and single shot diffraction patterns generated with high-current nanosecond-duration bunches, and 1000-shot registered-average diffraction using relatively low-current ultrafast bunches.

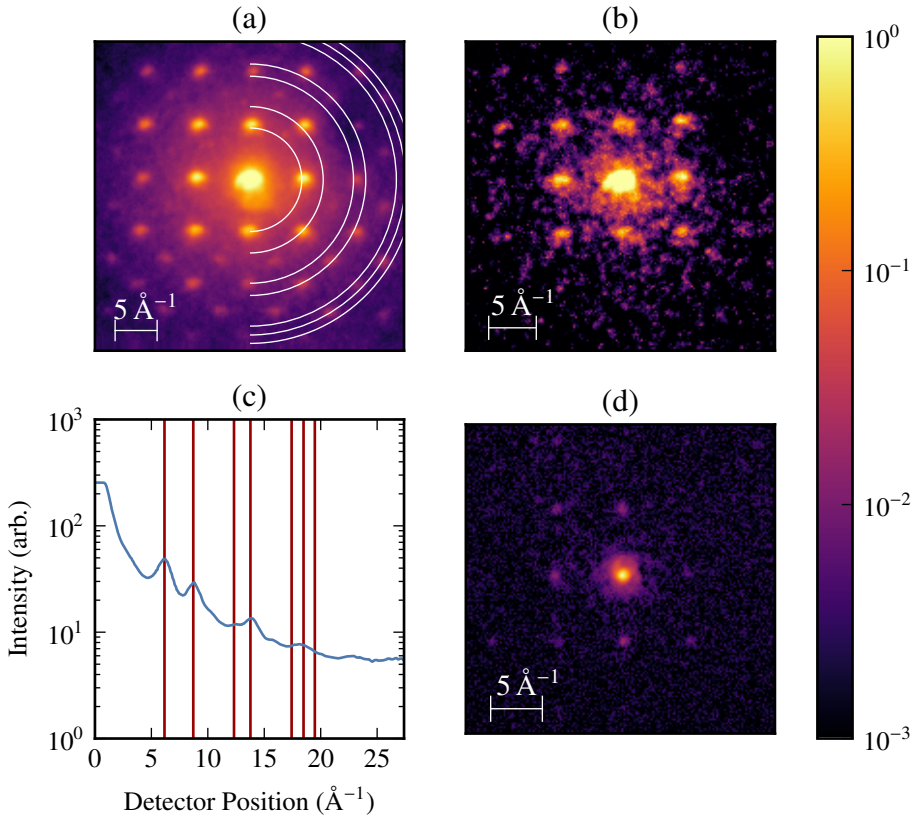


Figure 4.2: Log-scaled false-colour diffraction patterns from monocrystalline gold foil. (a) 100-shot average diffraction pattern generated with nanosecond-duration electron bunches. (b) Single-shot from the set that forms (a). (c) Radial average of (a). (d) 1000-shot registered-average diffraction pattern generated with ultrafast electron bunches. The lattice spacings for gold are indicated by white rings (a) and red lines (c). The colour scale for each image is individually scaled to optimise visibility. The colour bar indicates the log-scaled colour mapping for the normalised data.

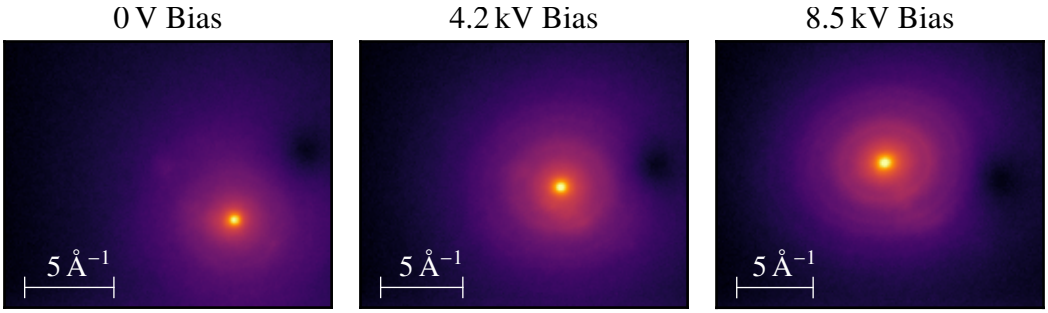


Figure 4.3: Log-scaled false-colour diffraction patterns from polycrystalline aluminium with an initial electron beam energy of 11.7 keV and a voltage bias of 0, 4.2 keV and 8.5 kV so that the effective diffraction energy was 11.7 keV, 15.9 keV and 20.2 keV respectively. The scale bars are calculated from the average radius of the innermost diffraction ring. The colour mapping is the same as that shown in Figure 4.2.

Ultrafast Diffraction from Gold

The ultrafast bunches were produced using two-colour multiphoton excitation, see Section 2.1.4, which was able to generate a few hundred electrons per shot. Due to the lower current of the of the ultrafast bunches the Bragg spots are only visible with multi-shot averages. A logarithmically-scaled 1000-shot average of ultrafast diffraction from gold is shown in Figure 4.2(d). Despite the low signal there was sufficient intensity in the central spot to enable the individual shots to be registered using the technique described in Section 2.3. A halo is visible about the central spot, generated from poorly focused electrons from single colour multiphoton excitation.

The electron bunches used to produce the data shown in Figure 4.2 were not themselves verified to be ultrafast via streaking but equivalent bunches were verified, as shown in Reference [46], and it is reasonable to assume that these bunches also had durations of tens of picoseconds.

This demonstration of ultrafast diffraction is an important milestone in the development of cold atom electron sources but significant improvements to ultrafast beam current are required before the CAES can perform diffraction that is both single-shot and ultrafast.

4.3.2 Aluminium

The CAES was also tested with a polycrystalline evaporated aluminium foil with a thickness of 31 nm[1].

The 11.7 keV electrons produced by the CAES underwent a significant amount of multiple scattering, such that the visibility of the diffraction rings was extremely low. The use of

[1] Diffraction Standard Evaporated Aluminium, Product No. 619, Ted Pella. Inc.

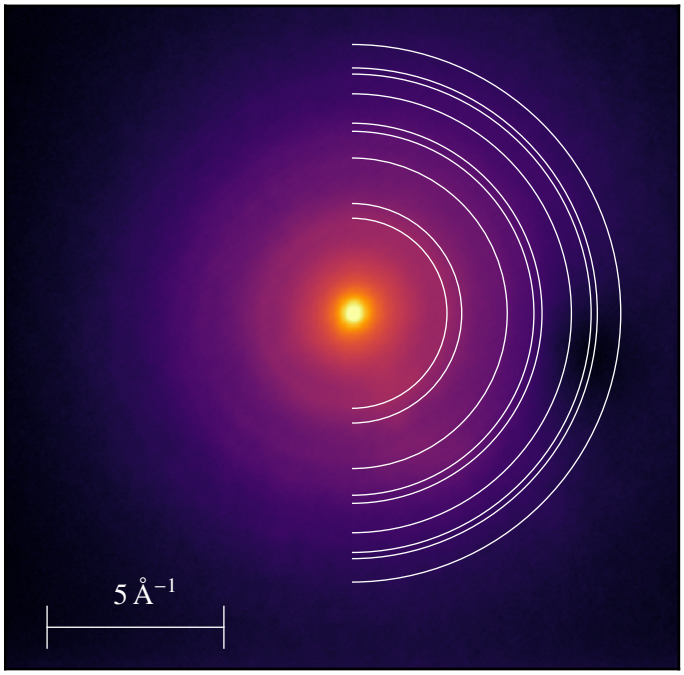


Figure 4.4: Log-scaled false-colour diffraction pattern from polycrystalline aluminium with an initial electron beam energy of 11.7 keV and a voltage bias of 8.5 kV so that the effective diffraction energy was 20.2 keV. The white rings indicate the expected locations of the aluminium diffraction rings. Due to the non-linear distortion from the sample voltage bias the rings do not match the expected locations. The ellipticity apparent in Figure 4.3 has been corrected in this image and the scale bar is calculated from the radius of the innermost diffraction ring. The colour mapping is the same as that shown in Figure 4.2.

the sample bias voltage allowed for higher energy electrons which reduced the incidence of multiple scattering such that the visibility of the aluminium diffraction rings improved. This is shown in Figure 4.3 where, as the sample bias is increased the visibility and separation of the rings improves. It was not possible to increase the bias voltage beyond 8.5 kV due to electrical breakdown. The fields generated by the sample bias also cause some displacement and distortion in the electron beam as can be see by the lateral shift of the diffraction pattern and slight ellipticity of the high-bias image.

In Figure 4.4 the high-energy aluminium data has been shown with the expected locations of the diffraction rings overlaid. The lower-order rings are close to the expected locations but the higher-order rings do not line up well due to the non-linear distortion to the post-sample electron trajectories caused by the strong field from sample bias voltage.

If a thinner aluminium sample was used or if the beam energy could be increased further then fewer multiple scattering events would occur and the signal-to-noise would be improved.

4.4 Conclusion

The CAES has previously been used to demonstrate single-shot diffraction as shown in Reference [1]. Here it has been shown, for the first time, that a CAES can be used for ultrafast diffraction, using a simple test case target of crystalline gold. These demonstrations of transmission electron microscopy are important steps along the road towards the many potential applications of the CAES.

The limitations in the current accelerator design have been highlighted and circumvented with the use of a bias voltage applied to the sample holder that allows for significant improvements to the visibility of the diffraction pattern from polycrystalline aluminium. Greater investments into the sophisticated accelerator technology already available to modern electron microscopes would allow future generations of cold-atom sources to access a much wider range of beam energies and for significantly more samples to be imaged. The next step is to demonstrate diffraction that is both single-shot and ultrafast which will require the next generation of CAES and the higher beam current it will be able to produce.

The impressive coherence and emittance of the CAES allows for greater brightness for a given bunch charge in comparison with other sources. With greater bunch current the CAES would be capable of demonstrating diffraction that is both single-shot and ultrafast. Higher beam current would also permit CDI and then the integration of rf-bunch compression would permit ultrafast molecular imaging. Short duration, high charge bunches would experience space-charge degradation of the beam quality and would therefore require the implementation of strategies to counter space-charge, taking advantage of the beam shaping capabilities of a CAES.

Chapter 5

Time-Resolved Brightness Measurement

The most comprehensive figure of merit for charged particle beams is brightness, which incorporates beam current and emittance [136]. Emittance can be improved, along with potential imaging resolution, with the aperturing of a beam but there will be a loss of beam current which will result in imaging tasks taking longer to complete. Brightness can be used to as a metric for both resolution and imaging time. Brightness measurements can also serve as a powerful diagnostic of the processes involved in beam creation and manipulation [137–139]. Brightness measurements are typically static but time-resolution has the potential to reveal information related to effects such as electron diffusion time, image-charge formation, and space-charge interactions. This chapter presents a technique for measuring time-resolved brightness by combining pepperpot emittance measurements with beam streaking, and demonstrates the method with electrons generated from the CAEIS This work has been published as Reference [57].

The pepperpot method measures the transverse emittance of a charged particle beam using an array of apertures or slits to divide the beam into smaller beamlets that are detected after propagation [140–143]. The divergence, and thus the emittance, can be determined by measuring the position and size of each beamlet, giving an approximation of the transverse phase space of the beam [142, 144]. The pepperpot method has been used on both electron and ion beams, and with electron energies up to 500 MeV [145].

Time-resolved emittance measurements have been performed previously using time-gated detectors to study the feasibility and performance of laser-generated ion sources [146, 147], and with scanning slits and the time-resolved signal from a Faraday cup with the aim of minimising the emittance of an electron storage ring [148]. These methods require many shots to accumulate the temporal profile, and consequently only measure the average beam behaviour and are fundamentally incapable of observing shot-to-shot variation in bunch dynamics. An alternative method to determine the temporal emittance profile of a bunch is to apply a time-varying deflection to the beamlets formed by a one-dimensional pepperpot mask (a line of apertures) to

'streak' the measurement across an imaging detector. Streaked measurements have been used in electron diffraction studies to observe non-repeatable dynamical processes [149, 150], and to characterise the electron pulse at the Linac Coherent Light Source (LCLS) [151]. With a sufficiently intense source, streak measurements could be performed in a single shot, providing information on shot-to-shot behaviour to elucidate information unavailable to multiple shot averages, such as transient and stochastic effects.

There are a number of important reasons for interest in time-resolved brightness measurements with a CAEIS, because high brightness is critical to achieving high spatial and temporal resolution imaging. Knowledge of source brightness is of particular interest in a number of areas such as the advantage of knowing beam coherence for UED, the ability to observe the performance of techniques to counter space-charge expansion, and the potential to provide information on the ionisation processes in a CAES. Beam emittance is also a vital factor when considering ion beam milling and ion microscope applications as the emittance limits the resolution achievable [58, 59].

The CAEIS is a promising candidate for UED imaging and structure determination due to its potential for high brightness and coherence, with the prospect of enabling the measurement of atomic-level sub-picosecond structural dynamics [32, 45, 152]. UED is often performed with photocathode electron sources which have sufficient current and bunch duration for UED with crystalline samples but has limited coherence [5, 8, 153]. A CAES has the necessary coherence for UED of nanocrystals and large molecules but not yet sufficient current [47, 76]. In the context of (ultrafast) diffractive imaging, time-resolved knowledge of the source coherence can be incorporated into the image reconstruction to improve the process and, as the coherence can be calculated from the emittance, the technique presented here could prove to be useful for achieving UED of nanocrystals and even non-crystalline targets [154, 155].

Electron bunches from the CAES can be produced with duration over a range of timescales, from femtoseconds to microseconds. With the use of femtosecond duration pulse lasers, a CAES can produce ultrashort electron bunches, which are dense enough to experience space-charge expansion and the related loss of beam quality [53, 55, 62]. One of the obstacles to single-shot, UED is degradation of beam quality due to space-charge expansion and streaked pepperpot measurements could be used to observe the performance of techniques designed to counter space-charge expansion [62].

The production of electrons from photoionised cold atoms is a complex process, and the temporal profile of pulses from these sources has been characterised under a variety of ionisation conditions [46]. The same complex ionisation processes that result in variable pulse duration are also likely to affect the transverse velocity of the emitted electrons, and thus the emittance, as a function of time. The streaked pepperpot method presented allows the transverse velocity spread of the liberated electrons to be measured as a function of time, which can illuminate the underlying atomic ionisation processes and provide a diagnostic which could

allow optimisation of electron bunch brightness in the future.

This chapter presents a simple method of measuring time-resolved emittance that is applicable to a wide range of charged particle sources and which could be used for single-shot measurement of sources with high currents. The streaked-pepperpot technique was developed to observe time-varying effects during the photoionisation and extraction of electrons from the CAES. The brightness of a CAEIS is important to potential applications and the time-resolution provided by the technique described in this chapter will be a useful tool to examine the behaviour of the source and hopefully provide avenues for improvement.

This chapter examines time-resolved emittance measurements through the streaking of pepperpots, including the theory involved, various technical considerations, and example measurements made with the CAEIS at the University of Melbourne.

5.1 Brightness and Emittance

A given ensemble of particles can be described by its density in six-dimensional phase space, (x, p_x, y, p_y, z, p_z) where (x, y, z) are the positions and (p_x, p_y, p_z) are the momenta of each particle. The extent of the beam in phase space is called the *emittance*, ϵ, of the beam. Each Cartesian direction is examined separately, (ϵ_x, x, p_x), (ϵ_y, y, p_y) and (ϵ_z, z, p_z) where z is the optic axis of the beam.

Typically the gradients of trajectories in x-z and y-z are measured rather than the momenta. These gradients are referred to as the divergence and are defined as

$$x' \equiv \frac{dx}{dz} = \frac{v_x}{v_z}. \tag{5.1}$$

The space of (x, x') is referred to as trace-space. An example of the trace-space occupied by a beam is shown in Figure 5.1. The *emittance* can be defined as

$$\epsilon_x \equiv \frac{A^x}{\pi} \tag{5.2}$$

where A^x is the area occupied by the beam in trace space.

The *RMS emittance* is a more practical measure and is defined as

$$\bar{\epsilon}_x \equiv \sqrt{\langle x^2 \rangle \langle x'^2 \rangle - \langle x x' \rangle^2} \simeq \frac{\epsilon_x}{4}. \tag{5.3}$$

The emittance of a beam represents the 'focusability' of the beam. A low emittance beam can be focused to a smaller waist than a high emittance beam, in fact a beam with zero emittance could be focused to a point whereas any beam with non-zero emittance has a finite spot size, this is shown in Figure 5.2. This makes emittance an important quantity to consider in almost all beam applications but is particularly important to charged beams due to complications produced by space-charge repulsion.

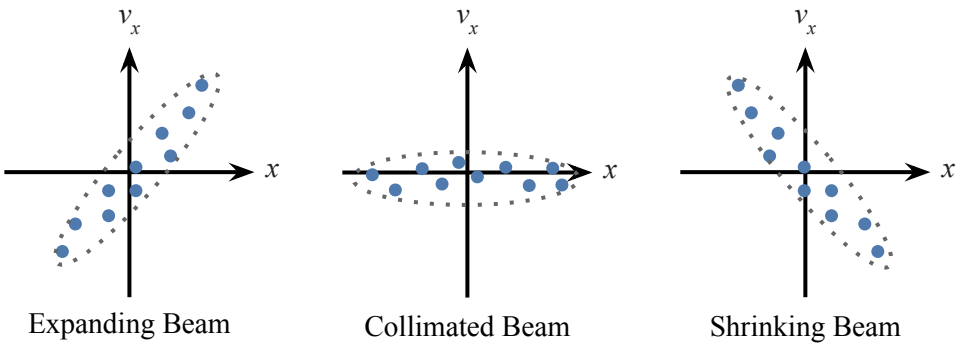


Figure 5.1: An example of the trace-space occupied by an expanding, collimated and shrinking beam. The dotted line indicates the area occupied by the beam, the emittance.

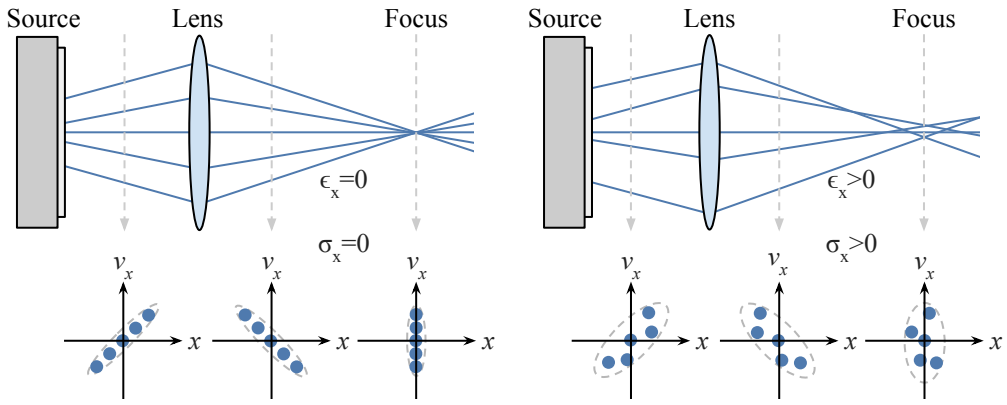


Figure 5.2: On the left is a particle beam with zero emittance which can be focused to a beam waist with zero width. On the right is a beam with non-zero emittance with a non-zero beam waist. Below each figure are plots of the phase space at difference points in the beam where the dashed line is indicative of the phase-space area or emittance.

The emittance of a beam is not the only factor that determines the quality of the beam. Emittance could be made arbitrarily small through the use of collimating slits however the reduced particle count would reduce the usefulness of that beam for most applications. A more comprehensive figure of merit is the number of particles with a given emittance, known as *brightness* [136];

$$B = \frac{I}{8\pi^2 \bar{\epsilon}_x \bar{\epsilon}_y}, \tag{5.4}$$

where I is the current of the beam and $\bar{\epsilon}_{x,y}$ is the RMS emittance as described above. Compared to emittance, which characterises the ultimate spatial resolution that can be obtained, brightness provides a better metric of beam quality because it incorporates the beam current which is important in determining the temporal resolution.

To compare the emittance and brightness of particle beams with different beam energy it is useful to define the *normalised* emittance and brightness

$$\bar{\epsilon}_{n,x} = \beta \bar{\epsilon}_x, \tag{5.5}$$

$$B_n = \frac{I}{8\pi^2 \bar{\epsilon}_{n,x} \bar{\epsilon}_{n,y}} \tag{5.6}$$

where $\beta = v/c \approx v_z/c$ and c is the speed of light. Normalised brightness and emittance are invariant for a particular beam under ideal conditions.

5.1.1 Emittance with a CAEIS

The normalised RMS emittance for a thermal source (such as the CAES), is given by [76]

$$\bar{\epsilon}_{n,x} = \sigma_x \sqrt{\frac{k_B T}{mc^2}} \tag{5.7}$$

where σ_x is the RMS beam radius, k_B is Boltzmann's constant, T is the temperature of the source, and m is the particle mass. Equation 5.7 highlights one of the advantages of a CAEIS: the low source temperature allows for low emittance bunches compared to common thermionic sources.

The temperature of the CAEIS source can be calculated from the wavelength of the ionisation lasers, λ_{red} and λ_{blue},

$$E_{red} = \frac{hc}{\lambda_{red}} \tag{5.8}$$

$$E_{blue} = \frac{hc}{\lambda_{blue}}, \tag{5.9}$$

where h is Planck's constant. We can define the total ionisation energy to be

$$E_{total} = E_{red} + E_{blue}. \tag{5.10}$$

The ionisation energy for Rubidium 85, E_I is 4.18 eV [119] so we can determine the excess energy of ionisation

$$E_{excess} = E_{total} - E_I \tag{5.11}$$

The excess energy can be related to the temperatures of the ionised particles via

$$T = \frac{E_{excess}}{k_B}, \tag{5.12}$$

if some of the details, such as disorder induced heating, are ignored. Combining Equations 5.7 and 5.12 allows for the calculation of the expected emittance of bunches generated from the CAEIS for above-threshold ionisation. The emittance of bunches generated from below-threshold pathways, such as Rydberg-excitation with field-ionisation, cannot be calculated using Equation 5.7, see Reference [50] for greater detail.

5.2 Measurement

Directly calculating the emittance of an ensemble with Equation 5.3 requires full knowledge of the position and momenta of the particles which is difficult because beam monitors tend to only measure the transverse positions of particles. There are a number of methods to measure the emittance of a particle beam, namely pepperpots, the multiple profile method and the examination of beam profile as the strength of a well characterised lens is varied. In the Melbourne CAEIS the lens method is not practical as the lenses in the system are not well characterised and multi-profile measurements are tedious due to the required manual z-translation of the detector and bellows. The pepperpot method however is achievable with this system although the details of the geometry are not ideal with this iteration of a CAEIS.

5.2.1 Pepperpots

The pepperpot method uses a beam mask, consisting of an array of apertures, to separate the beam into a number of 'beamlets' which are then propagated and imaged with a spatially resolved detector. By examining the size of the beamlets the divergence of the beam can be estimated and thus the emittance of the beam can be calculated. Ideally the extent of the array should be larger than the size of the beam and the holes as small as is practical while maintaining sufficient flux and ensuring the spots on the detector do not overlap. The name refers to the similarity of the simplest beam mask to the perforated lid of a container for pepper.

A useful derivation of this technique is presented in Reference [144] with the one dimensional result:

$$\begin{aligned}\epsilon_x^2 &= \left\langle x^2 \right\rangle \left\langle x'^2 \right\rangle - \langle x x' \rangle^2 \\ &\approx \frac{1}{N^2} \left\{ \left[\sum_{j=1}^{p} n_j \left(x_{sj} - \bar{x} \right)^2 \right] \left[\sum_{j=1}^{p} \left[n_j \sigma_{\bar{x}'_j}^2 + n_j \left(\bar{x}'_j - \bar{x}' \right)^2 \right] \right] - \left[\sum_{j=1}^{p} n_j x_{sj} \bar{x}'_j - N \bar{x} \bar{x}' \right]^2 \right\}\end{aligned} \tag{5.13}$$

where;

- N is the total number of particles after the beam mask,

- p is the total number of holes in the x direction,
- n_j is the number of particles passing through the j-th hole and hitting the detector,
- x_{sj} is position of the j-th hole,
- $\bar{x}$ is the mean position of the holes,
- $\sigma_{x'_j}$ is the RMS divergence of the j-th beamlet,
- $\bar{x'_j}$ is the mean divergence of the j-th beamlet, and
- $\bar{x'}$ is the mean divergence of all beamlets.

An example pepperpot mask and detected beamlets are shown in Figure 5.3. For two-dimensional pepperpots this equation can be implemented by appropriately rotating the detected beamlets and then performing row and column sums of the pixels followed by application of Equation 5.13 for x and y. For one-dimensional streaked pepperpot measurements the image should be rotated such that the streaks are horizontal, or vertical, and then each column, or row, can be used to determine the emittance at that point in the streak.

Temporal Resolution with Streaking

A one-dimensional pepperpot consisting of a line of apertures provides a good basis for streak measurements. In this case streaking is performed with a time varying electric field which deflects the charged particles across the detector. For a pulsed charged particle source, such as the CAEIS, the streak can be performed over the duration of the bunch or over a portion of the bunch if higher temporal resolution is required. There are some considerations for how fast an electric field can be swept, which will not be examined in detail here. Extremely short bunches will require more sophisticated streaking systems, such as an RF cavity or photoactivated switching which are able to provide temporal resolution as low as 100 fs [149, 156, 157]. An example of a streaked pepperpot measurement is shown in Figure 5.3d.

5.3 Experimental Setup

A number of modifications were made to the CAEIS for the streaked emittance measurements and there were a number of restrictions on various parameters due to the precise setup of the apparatus. The CAEIS was operated in electron mode for these measurements due to the available magnetic optics and the larger emittance expected from electrons compared to ions. The source temperature, and hence emittance, is higher for electrons, approximately 10 K for electrons and 100 μK for ions [47]. Using electrons also allows control of the bunch emittance as the excess energy can be controlled with the blue ionisation laser wavelength; with ions the change in emittance would be negligible.

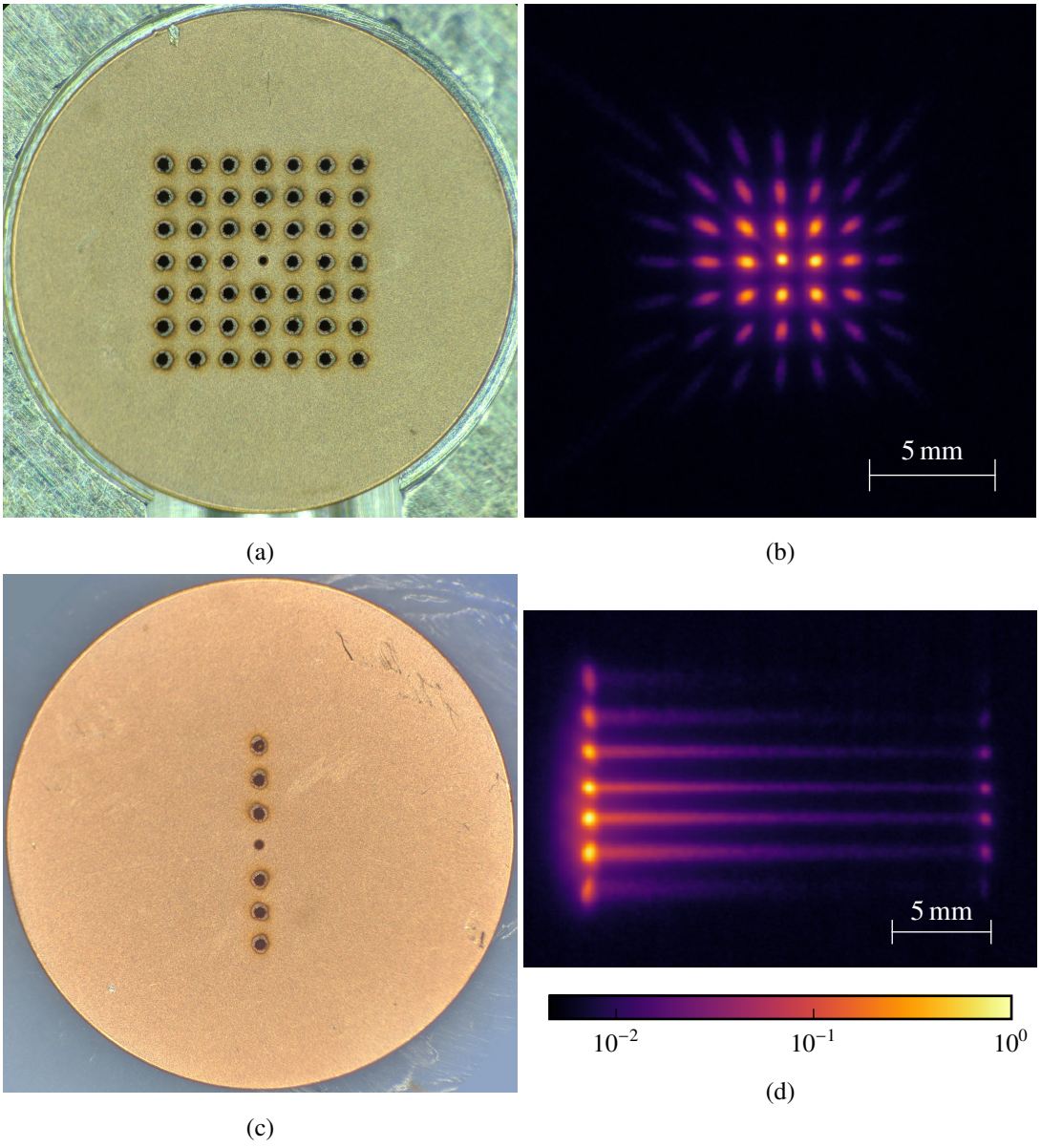


Figure 5.3: (a) a pepperpot mask cut into a thin 3 mm-diameter copper disk and, (b), the corresponding set of beamlets on the detector. (c) a one-dimensional pepperpot mask and the corresponding streaked set of beamlets on the detector, (d). The beam images are log scaled and the normalised colour scale is indicated in (d).

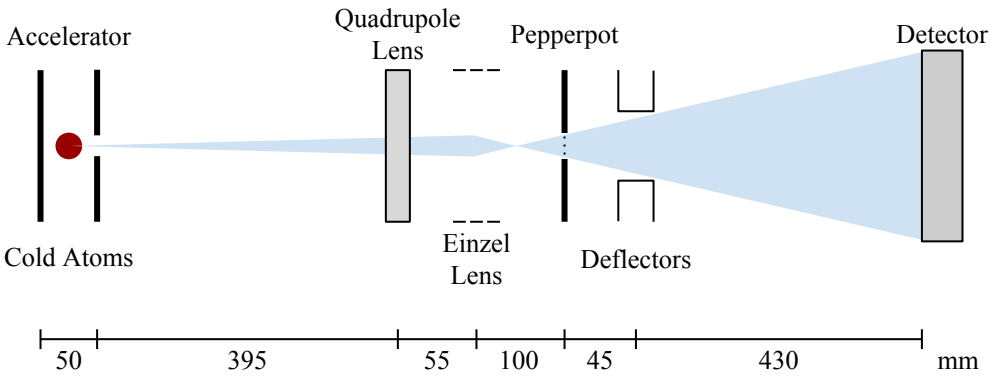


Figure 5.4: A schematic of the experimental apparatus with relevant dimensions. The blue region indicates the electron beam envelope. Note that components are not necessarily orientated realistically in this schematic (particularly the deflectors).

See Chapter 2 for a more detailed description of the CAEIS and Figure 5.4 for a schematic of the setup used for the measurements in this chapter.

5.3.1 Beam Optics

The quadrupole electron optics discussed in Section 2.4 were use to reduce the astigmatism present in the electron beam as discussed in the aforementioned section. The two-dimensional pepperpot provide another avenue to monitor the astigmatism of the beam because with an astigmatic beam the spacing of the beamlets on the detector are not the same along each axis as shown in Figure 5.5. The Einzel lens was used when focusing was required.

A number of permanent magnets were used to steer the beam through the various apertures and onto the detector and care was taken to keep the beam on the central axis of the apparatus. The static magnetic fields were configured manually by adjusting the positions of the magnets, which were mounted on posts external to the vacuum system. Due to the unstable nature of the magnetic environment and the multiple simultaneous research projects these magnets needed to be adjusted on a day-to-day basis to maintain the beam path.

The Einzel lens was used to focus the beam to a diameter approximately the same as the diameter of the pepperpot mask, as indicated in Figure 5.4. This focal arrangement was done to increase the divergence of the beamlets, making measurement easier, and to increase the electron flux transmitted through the pepperpot mask. The Einzel lens voltage required for this was approximately 4.8 kV and varied slightly with beam emittance, i.e. high emittance beams result in a larger beam size and thus required adjusting the Einzel lens voltage by ±100 V to keep the beam diameter equal to the pepperpot mask diameter at the mask.

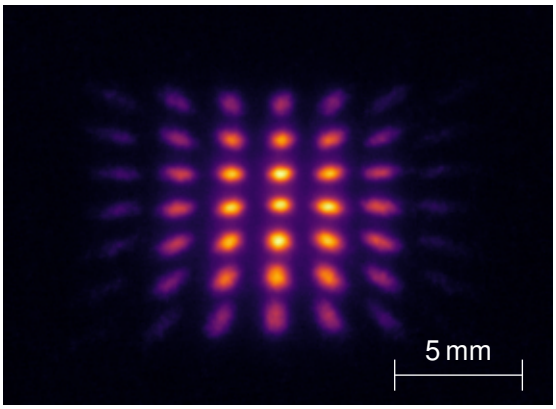


Figure 5.5: An example of the electron bunch after the pepperpot mask without the beam correction from the quadrupole lens. It is readily apparent that spacing of the beamlets is not the same for the x and y axes even though the spacing in the pepperpot mask were symmetric. This image is log scaled and the colour scale is the same as that shown in Figure 5.3.

5.3.2 Beam Energy

In many emittance measurements the beam energy would not be a free parameter. Emittance measurement would normally be used as a diagnostic for a specific system or scenario. In this experiment there were a number of considerations in determining the optimal beam energy to use for pepperpot measurements such as measurement resolution, beam current and beam stability and we were able to vary the energy of the beam from a few hundred eV to 11.7 keV.

The measurement resolution is affected by the beam energy. For a given emittance (transverse momentum spread), at low beam energy the size of the beamlets will be larger and thus more apparent at the detector than for high beam energy. Unfortunately the slower the beam the more fragile the alignment of the beam becomes with the CAES. It was impossible to align a 500 eV beam through the system to the detector and even had it been possible the stability of the beam would have been highly susceptible to the beam trajectory instabilities discussed in Section 2.2.3. A relatively high beam energy of 8 keV was used, providing robust beam alignment that was resistant to the transient changes in the magnetic environment and acceptable measurement resolution.

5.3.3 Bellows

The MCP detector could be translated along the z-axis of the beam as it was attached to the rest of the vacuum system with a set of vacuum bellows. The bellows provided an additional mechanism to control the resolution of the measurement by adjusting the propagation distance of the pepperpot beamlets and thus their size on the detector. Once the strength of the Einzel lens had been set as described above the bellows were positioned such that the length of the streaks covered the majority of the detector. The propagation distance from the pepperpot plan

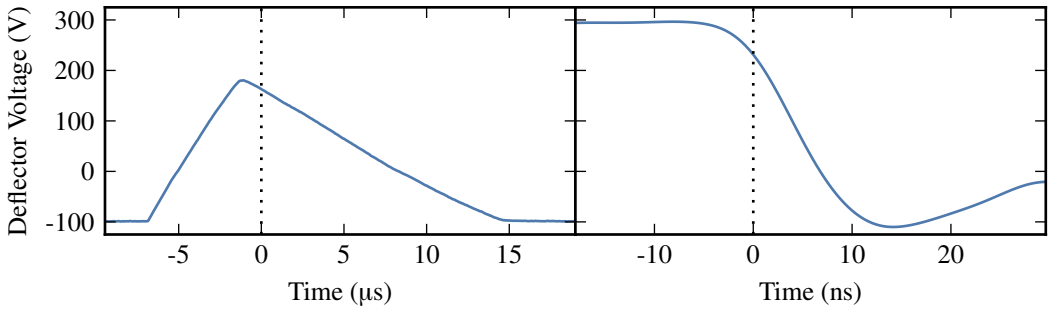


Figure 5.6: The potentials applied to the deflectors for the long duration (left) and short duration (right) bunches. The zero on the time axis refers to the start of the electron bunches as shown in Figure 5.17.

to the detector for this setup was 475 mm.

5.3.4 Streaking

The streaking was achieved using a pair of deflectors located after the pepperpot sample and aligned such that the one-dimensional pepperpots could be streaked across the detector. For these measurements one deflector was grounded and the other given a time-varying voltage which was supplied in one of two ways:

- a fast ramp using a bipolar push-pull solid-state switch with a fixed transition time of 10 ns and
- a slower ramp using an amplified signal generator with a minimum transition of approximately 10 μs.

An example of the voltage ramps is shown in Figure 5.6. The slow ramp was performed by amplifying the output of the signal generator[1], limited by the speed of the amplifier to a transition time minimum of approximately 10 μs in duration as shown in Figure 5.6. The streaking electronics were designed and constructed by fellow student Rory Speirs as described in Reference [71].

Synchronisation with the experimental cycle was achieved using triggers from the PulseBlaster, see Section 2.1.8, and, while the PulseBlaster did not have sufficient temporal resolution to correctly trigger the fast ramp precisely, adjusting cable lengths for the trigger signal by 1 to 2 m provided adequate control.

An interesting effect that limited the length of the usable portion of fast streaks was 'ringing' in the signal; shown in Figure 5.7. Due to the ringing the range of the voltage sweep is truncated because any bunch with a duration longer than the sweep time would oscillate about

[1] Rigol DG4162, bandwidth 160 MHz

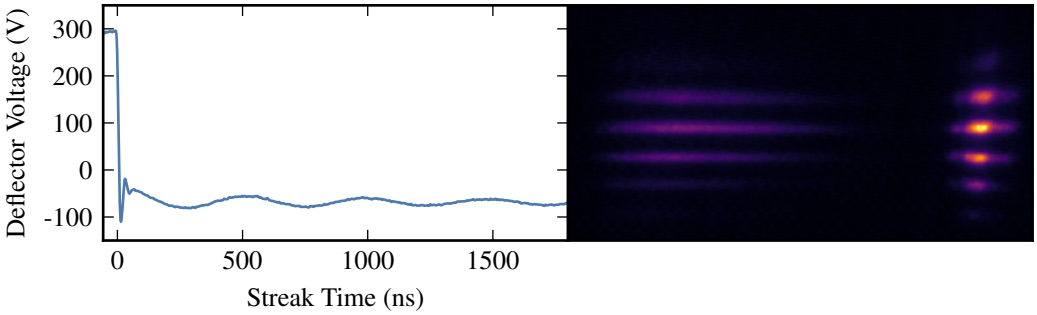


Figure 5.7: An example of ringing in the fast deflector voltage sweep. On the left is the voltage on the detector from the start of the electron bunch with a slow damping oscillation after the initial 10 ns sweep. On the right is an example of a streaked emittance measurement for a relatively long bunch length, showing oscillations corresponding to the 'ringing' present in the deflector voltage located on the right side of the streak. This image is log scaled and the colour scale is the same as that shown in Figure 5.3.

the end point of the streak. Fortunately the majority of the sweep is usable and bunches with duration shorter than the voltage sweep time are not affected.

Calibration

Calibration of the streak timing to the streak image was achieved with a calibration image, the voltage profile of the deflector during the streaking, and assuming the streak position-time relation was linear. The calibration image shows the electron bunch transmitted through the pepperpot mask at minimum and maximum deflection as shown in Figure 5.8. The calibration image allows for the determination of the deflection-voltage gradient, $\frac{dx}{dV}$. Any necessary transformations applied to the streaked pepperpot images, such as rotation and deskewing, were applied to the calibration image before measuring dx. The deflector voltage profile was measured with a 1 GHz oscilloscope[2] with a probe attached directly to the vacuum feedthrough. The voltage gradient, $\frac{dV}{dt}$, was measured by a linear fit to the streak region of the deflector voltage profile. The deflection gradient for the image can then be calculated as

$$\frac{dx}{dt} = \frac{dx}{dV}\frac{dV}{dt}. \tag{5.14}$$

The oscilloscope was also connected to a photodiode that monitored the intensity of the pulsed ionisation laser and this was used to determine the time during the voltage sweep that corresponds to the start of the streak. The beam current of the short duration streaks correlates strongly with the intensity of the blue laser as shown in Figure 5.9 which implies that the atom cloud is not saturated by the blue laser.

[2]Tektronix DPO4104B-L

Figure 5.8: An example of an image used to calibrate the timing of the streaked pepperpots. This image is composed of the average of several images where the deflectors were at the minimum and maximum voltages. This image is log scaled and the colour scale is the same as that shown in Figure 5.3.

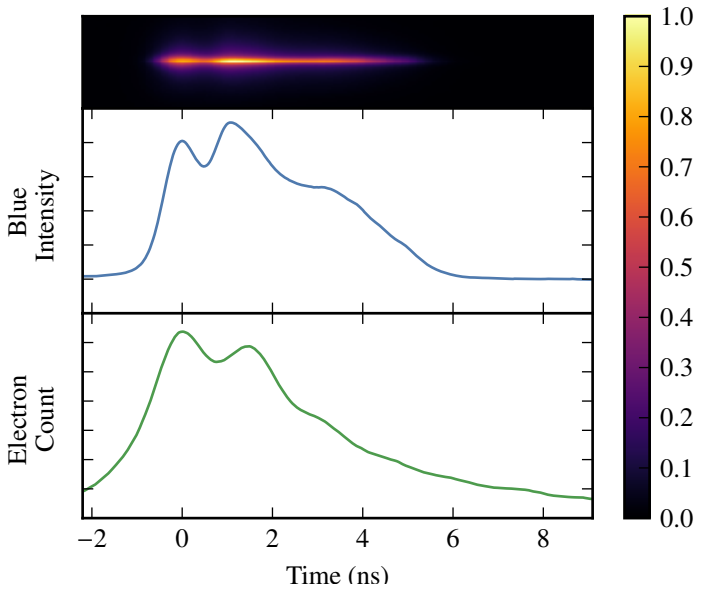


Figure 5.9: A comparison of the relative intensities of an unobstructed electron bunch focused and streaked across the detector and the blue laser pulse used to ionise the atoms. The top portion shows the streak of the electron bunch focused on the detector and streaked, the image is linearly scaled with a colour scale as indicated on the right. The middle portion shows the intensity of the blue laser, and the bottom portion shows the electron count calculated by summation of each column of the streak image.

5.3.5 Pepperpots

There were a number of iterations on the precise configuration of the one- and two-dimensional pepperpot masks used in these experiments. The considerations when designing pepperpot masks were:

- *Aperture size:* Smaller apertures provide better resolution to the emittance measurement but reduce the signal. The aperture size also affects the beamlet size and thus the potential beamlet overlap on the detector.

- *Aperture spacing:* Aperture spacing, also known as pitch, determines how well the full beam is sampled. Smaller pitch allows for better sampling but pitch should be large enough that the beamlet overlap on the detector can be corrected for, see Section 5.4.2. If the pitch is too small then the pepperpot can also become fragile.

- *Extent:* Ideally the total extent of the pepperpot should be much larger than the largest beam for a particular apparatus. Unfortunately, due to the mounting arrangement the size of samples was limited to 3 mm diameter and only a 2 mm diameter portion of the sample was accessible to the beam, see Figure 5.10.

There were a number of constraints on the pepperpot parameters. The maximum extent of the pepperpot masks were limited to the diameter accessible by the beam, 2 mm. The need to maximise flux, by focusing the beam to the pepperpot size, and avoid excessive beamlet overlap at the detector also limited the minimum feasible pitch. The pepperpots were machined[3] from 25 μm thick copper films[4] with 50 μm diameter holes. A pitch of 200 μm was chosen to provide acceptable sampling of the beam while minimising the overlap of beamlets on the detector. Given the 2 mm diameter available this pitch allowed for one dimensional pepperpots with 7 apertures and two dimensional pepperpots with 7×7 apertures. The pepperpots used are shown in Figures 5.3a and 5.3c.

5.3.6 Laser Parameters

The two-colour ionisation scheme required a CW red excitation laser and a pulsed blue ionisation laser, see Section 2.1.4.

Excitation Laser

The 780 nm excitation laser was frequency locked to the Rb85 cooling transition. The spatial profile of this laser was highly controllable with an SLM allowing for arbitrary profiles to be mapped to the electron or ion bunch [54]. The highest electron flux could be obtained by

[3]The pepperpots were cut using an Oxford Laser Systems Alpha 532 laser micromachining system.

[4]Gilder Grids GA50-C3, 50 μm aperture.

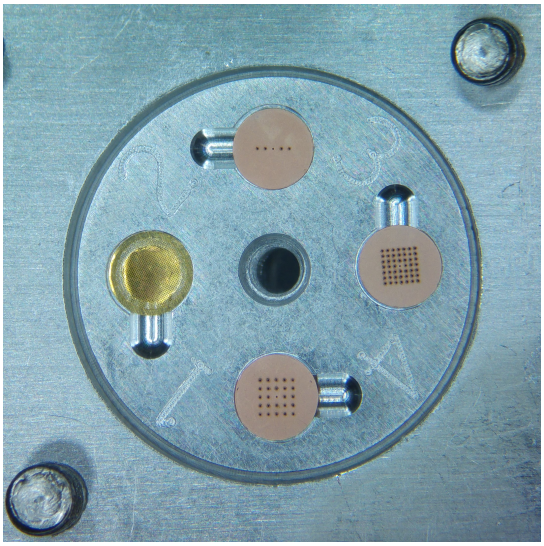


Figure 5.10: Photo of one of the sample holders that were used to load up to eight samples into the vacuum system. Clockwise from the top: five 50 μm apertures with 300 μm pitch, 7 by 7 pepperpot of 50 μm apertures with 200 μm pitch, 5 by 5 pepperpot of 50 μm apertures with 300 μm pitch, thin gold sample. Each sample is 3 mm in diameter. A 'lid' with appropriate apertures is placed over the samples to hold them in place leaving approximately 2 mm diameter of each sample accessible to the beam.

simply saturating the MOT but that limited control over the electron beam profile and size. One commonly used distribution is a flat-top which provided uniform electron intensity across the sample under examination. However if the electron beam emittance is high, as the flat-top beam propagates the beam profile will degrade to something closer to a Gaussian (see Figure 5.11).

If the electron beam has a simple, non-uniform spatial distribution, such as a Gaussian, then the specifics of the beam profile can be extrapolated from the pepperpot images allowing for further metrics to be calculated such as beam size and total electron count. A Gaussian distribution also maintains a similar profile with propagation even for high emittance beams. An example is shown in Figure 5.12. The rows and columns of the pepperpot image can be summed to generate a one-dimensional set of beamlets. From the amplitude and standard deviation of each beamlet the total number of electrons in each beamlet can be calculated and this corresponds to the number of electrons that passed through the corresponding aperture. A Gaussian can be fitted to the beamlet electron counts to approximate the full beam profile and the beam size and full beam electron count can be determined.

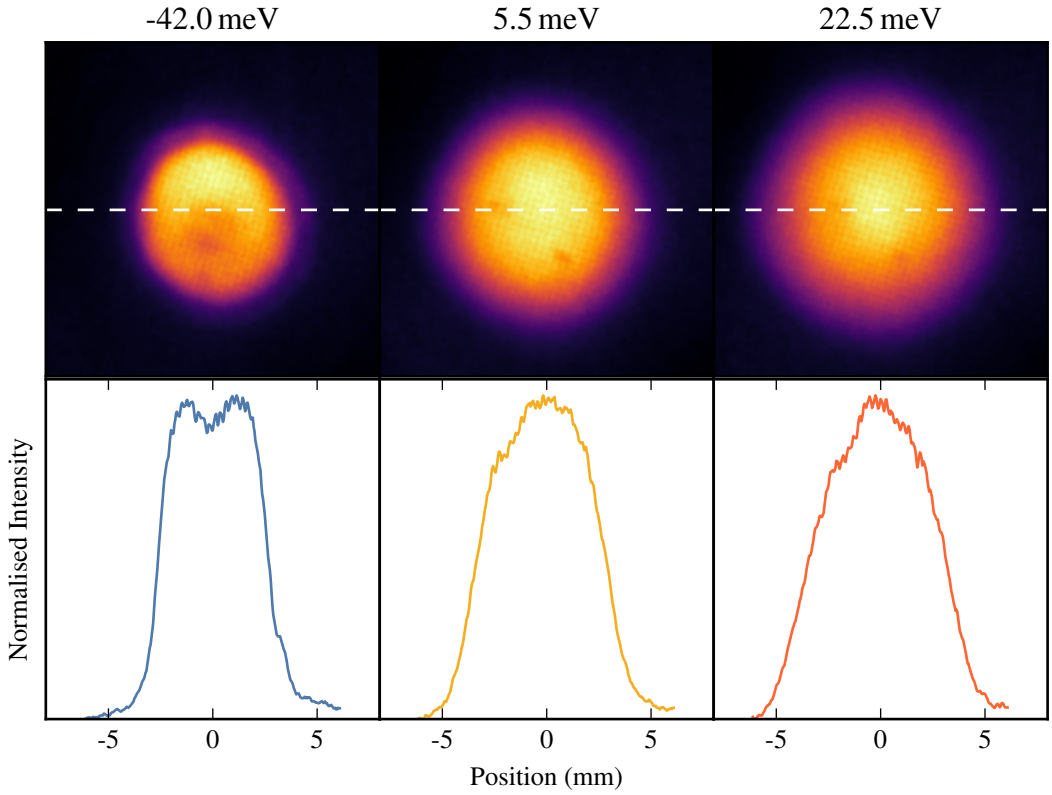


Figure 5.11: Images of electron beams that are unobstructed and not focused. The beams are generated with a flat-top profile on the excitation laser with varying ionisation laser wavelengths, and thus excess energies. The excess ionisation energy is listed above each beam profile. Below each profile is a trace of the central row of pixels (indicated by the dashed white line). The images are linearly scaled with and the colour scale is the same as that in Figure 5.9.

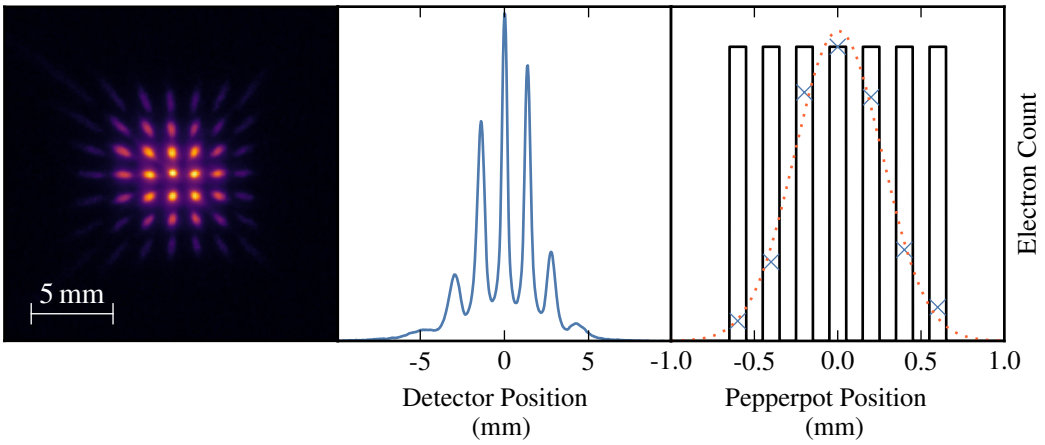


Figure 5.12: Left: image of electrons propagated through a 7×7 pepperpot with a below-threshold low-emittance electron bunch. Centre: column sum of the pixels in the first image for which the amplitude and standard deviation are calculated for each of the seven peaks. Right: black line indicates the shape of the pepperpot in one-dimension, the blue crosses indicate the total number of electrons in each of the seven, column-summed, beamlets and the red line indicates a Gaussian fitted to the beamlet electron counts and represents the full profile of the electron bunch incident on the pepperpots.

Ionisation Laser

A 457 to 493 nm 5 ns pulsed laser[5] was used to ionise atoms in the 5P excited state. The wavelength could be tuned to select various ionisation pathways such as above-threshold ionisation or field ionisation. The ionisation pathway selected affects the duration of the bunch produced. Above threshold ionisation resulted in short bunches with duration the same as the duration of the ionisation laser and below threshold ionisation in much longer bunches with duration of 10s µs due to electrons tunnelling out after the end of the ionisation pulse.

The ionisation laser wavelength also affected the excess electron energy and transverse momentum spread and thus the emittance of the electron bunches. This is discussed in Sections 2.1.4 and 5.1.1.

5.4 Simulations

Due to experimental constraints including the low beam flux, pepperpot mask size and beamlet overlap it was not possible to engineer an ideal setup of beam and pepperpot parameters. The non-ideal experimental parameters cause some distortions to the emittance measurements so simulations were performed to explore the effects and verify the validity of the corrections

[5] Sirah dye laser system CSTR-D-3000.

for these effects. These simulations also proved to be a useful confirmation of the analysis procedure used on the experimental data.

The simulations were performed using simple homemade particle tracking code. The code allowed for arbitrary bunch profiles and pepperpot masks defined in one and two dimensions. The code of the simulations is given in Appendix B.

A Gaussian bunch profile was used and the emittance of the bunch was set to a range of values similar to those achievable with the actual apparatus. The path of the bunch was designed to replicate that of the CAEIS by propagating the bunch to a lens, then to a pepperpot mask and finally to a detector. The lens was implemented by applying a radially dependent velocity change to the particles in order to focus the beam such that the beam size was the same as the size of the pepperpot mask, approximately 2 mm.

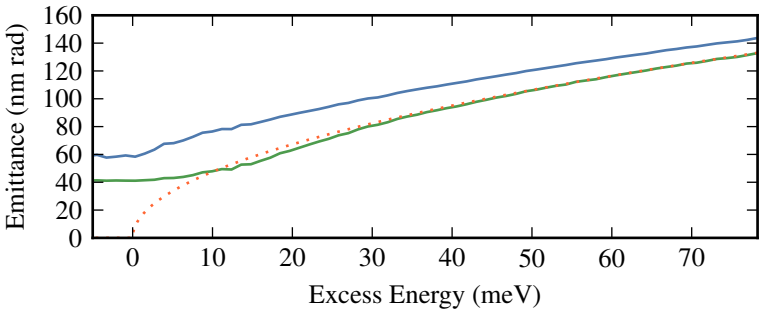
5.4.1 Pepperpot Aperture Size

The first set of simulations investigated the effects of aperture size on the emittance measurement. The procedure outlined in Reference [144], which formed the foundation of the analysis used here, assumes that the size of the apertures used is negligible compared to the divergence of the particles. Due to the low electron flux it is not feasible to use pepperpots with negligibly small holes so it is necessary to understand and correct for finite aperture size.

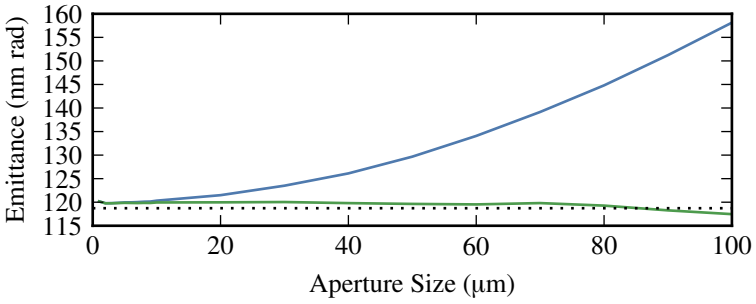
The profile of a beamlet is a convolution of the aperture profile and the beamlet profile from an infinitesimally small aperture. Thus, when the aperture size is large compared to the effects of the beam divergence, it will be difficult to determine the divergence whereas if the divergence effect is similar or large compared to the aperture size then the divergence will be easier to determine. This results in an overestimation of beam divergence when the divergence and aperture size are similar and a minimum measurable divergence, and thus emittance, if the divergence is much smaller than the aperture size. An example of a resolution limit for this experiment is shown in Figure 5.13a.

The correction for finite apertures, where the aperture is not too large compared to the effect of the divergence, is to deconvolve the aperture size from the beamlet profile which is most easily implemented by fitting the width of a Gaussian convolved with the aperture size to the known beamlet size, taking magnification into account. Reapplying the magnification to the result of the fit supplies the profile of beamlets had the apertures been infinitesimally small.

The results of the analysis of simulations that vary the size of the apertures is shown in Figure 5.13b, and the improvement in the accuracy of the analysis when the aperture size correction is applied is evident.



(a) Simulated emittance measured as excess ionisation energy is varied for the electron bunch. The pepperpot mask had 50 µm diameter apertures with an extent much larger than the beam size and there was negligible beam overlap. The blue and green lines are the results without and with aperture size correction. The red line is the theoretical emittance given by Equation 5.12. The resolution limit of the mask is apparent with excess energies less than 10 meV.



(b) Simulated emittance measured as the size of the pepperpot apertures is varied. The blue and green lines are the results without and with the aperture size correction. The dotted black line is the true emittance of the bunch being simulated.

Figure 5.13

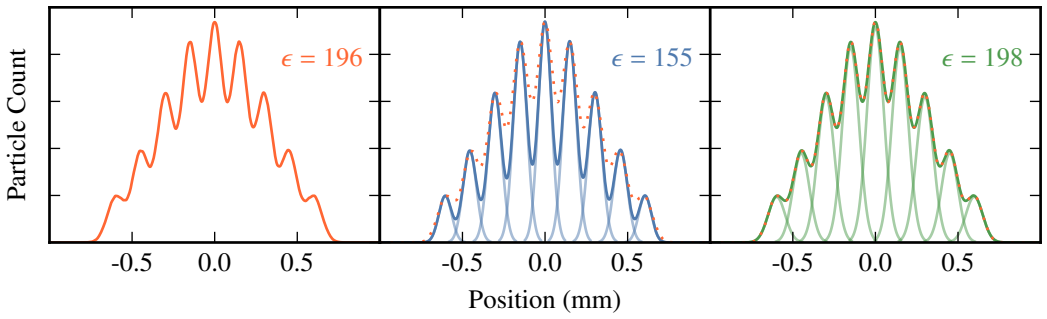


Figure 5.14: Simulated one-dimensional pepperpot emittance measurement, demonstrating overlapping beamlets. The left figure shows overlapping beamlets from simulated one-dimensional data (shown as a dotted line in the other figures), the middle figure shows nine Gaussian fits to each beamlet, and the summation of those fits, derived by considering only the portion of each beamlet between the troughs in the distribution, and the right figure shows nine Gaussian fits from fitting the sum of nine Gaussians to the data. The actual emittance of the simulated bunch is listed in the left figure and the emittance derived from the Gaussian sets is listed in the other two figures. In the second two figures the pale lines indicate individual Gaussians fitted for each beamlet and solid lines the sum of the Gaussians.

5.4.2 Beamlet Overlap

Most pepperpot analysis procedures assume that the beamlets are completely separated which was difficult to achieve with this apparatus while providing sufficient flux for streaked measurements and enough sampling of the beam to allow for estimation of the full-beam profile. A naïve analysis of overlapped beamlets might just analyse the potion of the beamlet between the troughs in the signal, but that results in an underestimate of the beamlet size due to the portion overlapping the neighbouring beamlets. The method used here is to fit the sum of N Gaussians, where N is the number of beamlets, to the data.

Shown in Figure 5.14 is an example of overlapped beamlets, and the improvement in the emittance measurement from the naïve approach to the fitting is clear in this example. The results of the naïve approach can be used as a guess for the fitting of the more detailed method. Most of the measurements made during this project have a lower degree of overlap than that demonstrated in Figure 5.14 and have a much smaller deviation from the correct emittance but the correction was still made to provide a small improvement in the accuracy of the analysis.

5.4.3 Pepperpot Extent and Beam Coverage

A pepperpot mask that has an extent smaller than the size of the beam is sampling only a portion of the beam and provides an underestimation of the beam emittance.

The emittance of a beam depends on the beam width (see Equations 5.3 and 5.7). If for

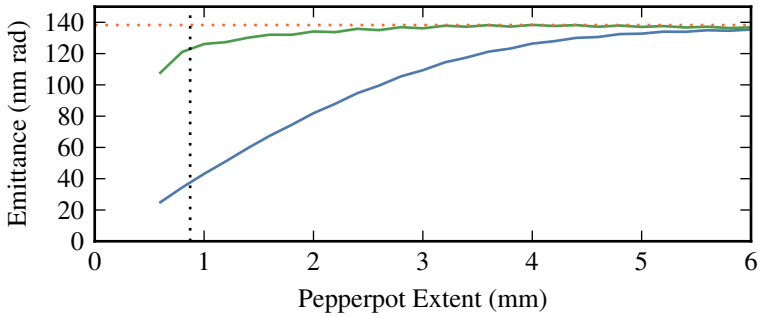


Figure 5.15: Simulated emittance as the extent of the pepperpot mask coverage of the beam is varied. The red dotted line indicates the true emittance of the bunch, the blue line indicates the measured emittance of the bunch without the beam coverage correction and the green line is the measured emittance with the correction. The dotted black line indicates the RMS width of the beam at the pepperpot plane (873 µm).

example, a pepperpot measures only half of a flat-top uniform distribution beam at a beam waist, then the measured emittance will be half the full emittance of the beam. The correction is more complicated for a Gaussian profile beam as the emittance is proportional to the integral of the normalised Gaussian covered by the pepperpot. If the beam shape in the pepperpot plane can be described by a Gaussian function $f_{\bar{x},\sigma}(x)$ then the true emittance should be

$$\epsilon_t = \epsilon_m \int_{-E}^{E} f_{\bar{x},\sigma}(x)dx, \tag{5.15}$$

where ϵ_m is the measured uncorrected emittance and the pepperpot extends from $-E$ to $+E$.

If the pepperpot mask measuring the beam is not at a beam waist then, due to the beam divergence, the correction shown in Equation 5.15 is not suitable. Adding another factor k to the limits of the integral in Equation 5.15 allows for a simple method of taking the divergence into account,

$$\epsilon_t = \epsilon_m \int_{-k\,E}^{k\,E} f_{\bar{x},\sigma}(x)dx. \tag{5.16}$$

The value of k depends on the divergence of the beam which in turn depends on the precise geometry of the measurement. Fitting to the simulations, which replicate the geometry of the apparatus used here, indicates that for this apparatus $k = 0.75 \pm 0.02$.

The effect of the correction is shown in Figure 5.15. The corrected emittance agrees well with the known emittance for the beam provided the extent of the pepperpot array is larger than the RMS beam width. The extent of the pepperpots in the measurements described in the following sections were much greater than the RMS beam width at the pepperpot plane.

5.5 Brightness Measurements

To demonstrate the feasibility of time-resolved brightness measurements a number of measurements have been made including a comparison of two-dimensional pepperpot emittance measurements with theory and simulation, and one-dimensional pepperpot streaked measurements with long and short duration electron bunches.

5.5.1 Analysis

The analysis procedure developed was based on the theory described in Reference [144] and Equation 5.13. The core of the code for the analysis is given in Appendix B. To determine the emittance from a one dimensional pepperpot measurement a number of parameters are required:

- Size of pepperpot apertures
- Position of pepperpot apertures
- Size of beamlets on the detector
- Position of beamlets on the detector
- Total number of electrons in each beamlet

The pepperpot parameters are known from their fabrication and the beamlet parameters can be determined from appropriately prepared two-dimensional and streaked one-dimensional pepperpot measurements. The analysis procedure is as follows:

1. Acquire electron projection images
2. Register images (see Section 2.3)
3. Average registered images
4. Rotate image so dominant features are aligned with the pixel rows
5. Deskew image so features along the second axis align with the columns
    - For two dimensional pepperpots, sum the rows and columns separately to produce two one-dimensional sets of beamlets
    - For streaked emittance measurements, consider each column of pixels independently as a one-dimensional sets of beamlets.
6. Determine the approximate amplitude, mean position and RMS width of each beamlet

7. Refine the amplitude, mean position and RMS width of each beamlet by fitting a number of Gaussians equal to the number of beamlets to the beamlet set to account for any overlap

8. Transform the beamlet position and width measurements from pixel measurement to detector-plane measurements using the camera and MCP calibration

9. Fit a Gaussian convolved with the aperture size to the refined beamlet parameters in order to determine the equivalent beamlet for an infinitesimal aperture

10. Use Equation 5.13 to determine the emittance

11. Estimate the full-beam profile of the bunch by fitting a Gaussian to the number of particles transmitted through each aperture (equal to the total particle count for each beamlet)

12. Correct the emittance by considering the beam coverage of the pepperpot mask given the full-beam profile

13. Use the beam profile to determine the beam current

14. Use the beam current and emittance to determine the brightness

5.5.2 Calibrations

A number of experimental parameters require calibration and the procedures for doing so are briefly outlined here.

Detector-Camera Distance Calibration

The calibration of camera pixel count to real distance in the electron imaging plane was 40.8 μm per pixel, determined from an image of a standard ruler placed beside the phosphor screen of the MCP detector.

Detector Efficiency

The number of camera counts generated by a single electron was determined by focusing a beam on to the Faraday cup and measuring the number of electrons per bunch, and recording an image for the same current in a defocused beam. The total number of counts on the image can then be compared to the known number of electrons in the bunch. To reduce the uncertainty in this measurement, it was repeated for several different electron currents. The calibration was found to be 34.3 counts per electron.

This process was performed with the MCP voltage at 1.7 kV and the phosphor voltage at 4.0 kV, the values used for all measurements in this chapter.

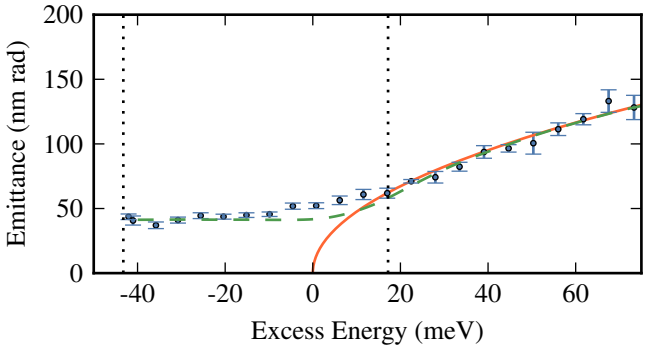


Figure 5.16: Emittance of electron bunches produced by the CAES (blue circles) compared to theoretical predictions of Equation 5.7 (solid red line), and the results of analysis of simulation (dashed green line) as the excess energy is varied. The black dotted lines indicate the excess energy of the two streaks shown in Figure 5.17.

Source Size

A precise calculation of the size of the electron bunch at the source would require the convolution of numerous measurements; the size and profile of the two ionisation lasers at the MOT, and the spatial profile of the MOT combined with the non-linear atom-light interactions. This complex calculation of source size is beyond the scope of this thesis but the source size can be estimated with fitting Equation 5.7 to the data presented in Figure 5.16 and discussed in next section. By fitting Equation 5.7 to the data shown in Figure 5.16 it was apparent that the source RMS size was 340 μm, comparable to the size given for this apparatus in Reference [1].

5.5.3 Two-Dimensional Pepperpots

The excess energy and hence emittance of the electron bunches could be controlled through variation of the ionisation laser wavelength. Varying the wavelength of the ionisation laser and measuring the emittance of the electron bunches using the two dimensional pepperpot allowed for testing of the emittance measurement system and analysis as the results could be compared to the theory described by Equation 5.7 and the simulations in Section 5.4.

The results from this test can be seen in Figure 5.16 and the results are well matched to the theory and previous measurements of the emittance of this CAES [76]. The data are taken from 100-shot registered averages and agree with the simulation results. The lower bound on measurable emittance is apparent in both the measurement and simulation results. Due to the aperture size of 50 μm and beam path geometry (propagation distances, focusing parameter, etc.) the measurements were limited to resolving emittance greater than 41 nm rad.

5.5.4 Streaked One-Dimensional Pepperpots

Shown in Figure 5.17 are examples of time-resolved brightness measurements from streaked one-dimensional pepperpots. The streaks were formed from long- and short-duration electron bunches produced from the CAES with below- and above-threshold ionisation and the images created from 1000-shot registered averages. The long-duration bunch was generated with a blue ionisation laser wavelength of 487.2 nm corresponding to an excess ionisation energy of −43.25 meV, below-threshold ionisation and a bunch duration of order 10 µs. The short-duration bunch was generated with a blue wavelength of 475.9 nm with an excess energy of 17.18 meV, resulting in above-threshold ionisation and a bunch duration of 5 ns. These wavelengths were carefully chosen to provide good demonstrations of long- and short-duration bunches.

Temporal Resolution

The temporal resolution of these measurements depends on the point spread function (PSF) of the detector, the gradient or 'slew' of the deflector voltages and the size of the beamlets on the detector. The CCD camera and lens used to take images of the detector is assumed to have a resolution much better than the point spread function of the phosphor screen.

The PSF of the MCP and phosphor screen is approximately Gaussian with a standard deviation of 35 µm, measured by examining single electron events on the detector [71]. The streak calibration images (such as Figure 5.8) can be used as a simple measure for the size of the detected beamlets, which are a convolution of the beamlets size at the detector and the PSF of the detector. The detected beamlets tend to have an RMS width of 200 µm although this does vary with the emittance of the beam. The majority of the signal from an electron is therefore contained within an area with radius four times the standard deviation or 800 µm. The long- and short-duration streaks shown in Figure 5.17 can be sliced into 27 and 18 temporal slices where each slice is 800 µm wide and represents an independent temporal slice. The long- and short-duration streaks therefore have time resolutions of 524 ns and 247 ps respectively.

Ensuring that the streak takes up the maximum available extent on the detector minimises the effect of the PSF on the temporal resolution with the downside that signal is now spread over a wider area thus reducing the SNR.

5.5.5 Electron Flux

The electron flux of the CAES is not sufficient to perform the streaked emittance measurement in a single-shot and in some scenarios (for example when using ionisation pathways with poor coupling) the signal is so low that registration of measurements cannot be performed.

There are a number of strategies available to the CAES to improve the number of electron per shot such as increasing the MOT density, optimising the ionisation pathways, and using a

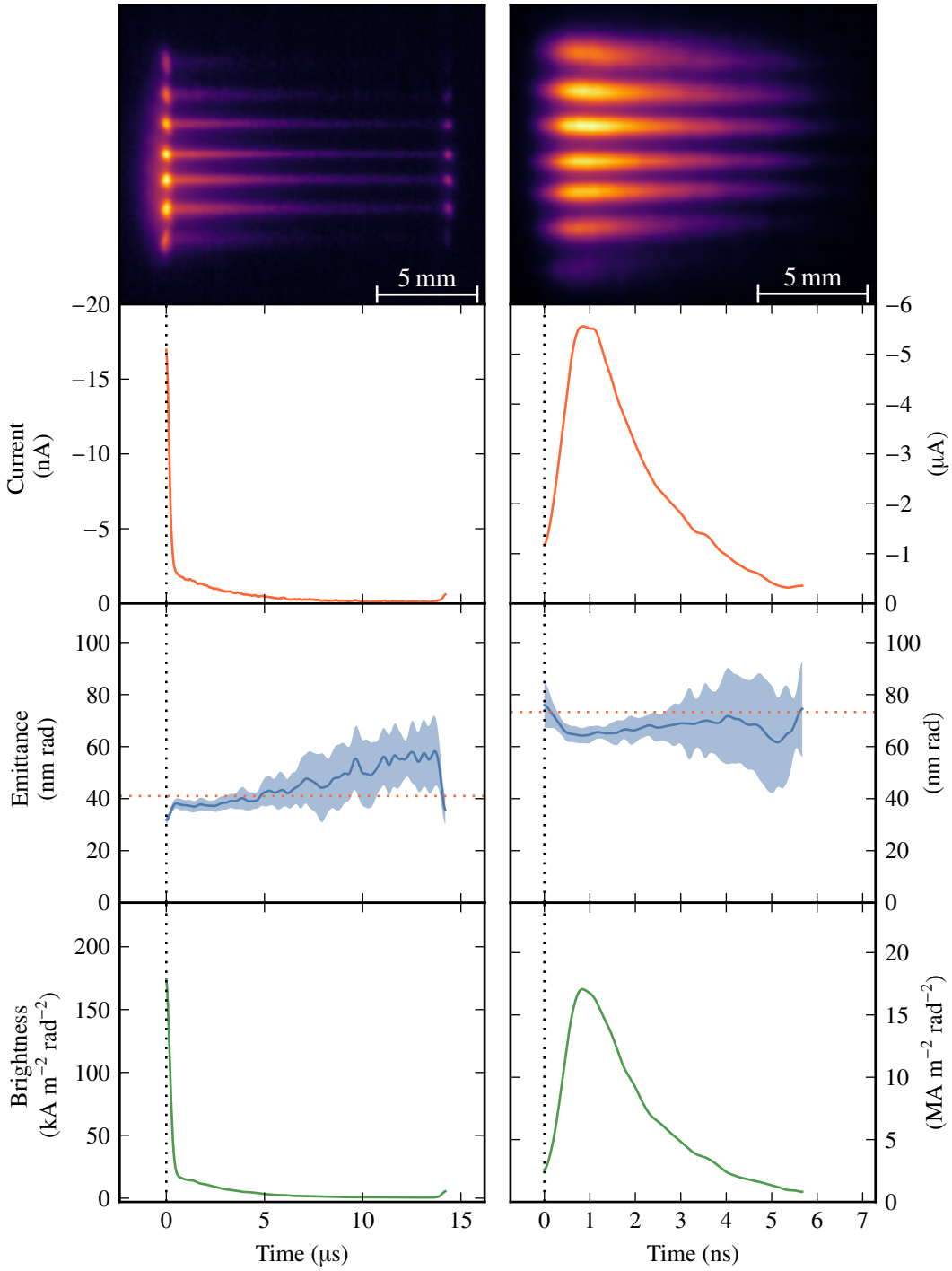


Figure 5.17: Long (left) and short (right) duration electron bunch streaked pepperpot measurements showing, from top to bottom, the log-scaled streak image (colour scale as in Figure 5.3), full-beam current, normalised RMS emittance, and normalised brightness. The dotted black lines indicate the start of the electron bunches. The red dotted lines in the emittance measurement indicate the expected emittance from the simulations discussed in Section 5.4. The shaded regions indicate the standard deviation of ten 100-shot measurements.

higher intensity blue ionisation laser. Certain photocathode sources are able to achieve electron counts of order 10^8 electrons per bunch and the streak method described here could achieve good emittance measurement in a single shot with that source and thus prove useful for beam diagnostics [149, 158].

5.6 Conclusion

This chapter has presented the first demonstration of time-resolved brightness measurements using streaked pepperpots. The technique was developed to observe time-varying effects during the ionisation and extraction of electrons from a CAES but the results show that the emittance over the duration of the bunches is constant, indicating that, for the conditions investigated here, there are no time-dependent effects on source emittance, such as hotter electrons produced through alternative ionisation pathways like multi-photon ionisation [46]. The measured emittance of the bunches correlates well with the results of simulation and theory, though the the below-threshold bunch measurements were limited by the minimum resolvable emittance of the apparatus. The temporal resolution could be enhanced by carefully maximising the length of the streak on the detector, using faster sweeps on the deflectors, or by using alternative deflection methods such as rf-cavities or photoactivated switches which are able to achieve 100 fs resolution [149, 156, 157].

This streaked pepperpot technique has promise for use in a wide range of charged particle beam sources and scenarios. With a sufficiently high-current source, such as a photocathode electron source, this technique could also be used to perform measurements in a single-shot. With the next generation of CAEIS currently under development this technique could be refined with less constrained apparatus, provide interesting insights into the ionisation processes involved and provide insights towards developing a source capable of UED. Time-resolved brightness measurements through the streaking of one-dimensional pepperpots have the potential to provide information on the behaviour of many charged particle sources, allow for a better understanding of the physics involved and suggest pathways towards improvement to these sources.

Chapter 6

Conclusion

A number of experiments have been performed with the University of Melbourne cold-atom electron and ion source (CAEIS) with a view towards an ultimate goal of single-shot, ultrafast coherent diffractive imaging (CDI) of single molecules. The research detailed in this thesis describes the final set of research performed with the University of Melbourne CAEIS, establishing an improved understanding of critical performance factors which will help to guide development of a future CAEIS.

In Chapter 3 laser frequency stabilisation using polarisation spectroscopy (PS) was demonstrated with impressive linewidth reduction, narrowing the linewidth of a Littrow configuration external cavity diode laser from several hundred kHz to 600 Hz and that of a cat-eye filter laser to 360 Hz. Sub-kilohertz linewidth is two orders of magnitude lower than previously shown with PS and previously only achieved with optical or electronic feedback from high-finesse optical cavities. The long-term frequency drift had a standard deviation of 51 kHz over a 60 hour period, more than adequate for most laser cooling experiments and again much lower than previously reported. Even greater improvements to laser frequency linewidth and drift are expected if feasible reductions in the noise are implemented.

The CAEIS was originally intended to be a high-brightness electron and ion source. The finely controlled ionisation method allows for lower source temperatures and thus higher brightness compared to those achievable with thermionic sources. The low emittance of the CAEIS has been demonstrated multiple times along with the 10 K source temperature, which is much lower than the temperatures typical of electrons originating from solid photocathodes.

Cold-atom electron sources (CAESs) are progressing well along the path towards single-shot ultrafast CDI but there is still a long way to go. The CAEIS described in Chapter 2 was capable of single-shot diffraction through monocrystalline gold and ultrafast diffraction (see Chapter 4) through gold. The beam current of the device was not sufficient to achieve diffraction measurements that were both single-shot and ultrafast. In Section 2.2 some strategies for increasing the beam current were investigated: continuous source operation, and the effects of rubidium oven temperature and ionisation laser intensity on the beam current. These inves-

tigations indicated that the main limiting factor with this system was the power of the blue ionisation laser system, which was insufficient to completely ionise all exposed cold atoms. A blue laser with enough power to easily saturate the ionisation process would go a long way to providing the flux for CDI and single-shot, ultrafast diffraction.

The electron beam profile from the source was also investigated and improved with the characterisation and correction of astigmatism described in Section 2.4. Improvements to beam quality allow for the maximisation of signal and have similar effect to improving the beam current. Further improvements to the beam quality can be made with more sophisticated beam optics which fortunately have already been developed within the mature fields of electron and ion microscopy.

Instabilities in the electron trajectories were investigated in Section 2.2.3. Electron beam drift was a significant issue complicating the measurements presented in this thesis and the registration technique described in Section 2.3 was essential to working around the beam drift and extracting signal from the multi-shot diffraction and emittance measurements. The beam drift had previously been attributed to the fast switching of the high-current magnetic coils for the magneto-optic trapping but operating the source with continuous rather than pulsed ionisation revealed that the absence of the coils had little effect on the observed beam drift. The beam drift could be attributed to electric and magnetic fields caused by the high-power switches, step-down transformers and the steel of the vacuum system. The effect of the unstable beam trajectory was exacerbated by the long distance from the source to the detector. The sub-optimal design of the apparatus is partially the result of the general purpose design that allows for a wide range of investigations from atom-laser interactions and high-precision spectroscopy to electron and ion beam experiments. Many lessons have been learned during the years of design and operation of this CAEIS and if it was to be redesigned from scratch then there are many possible improvements to minimise or eliminate beam drift, such as a shorter propagation distance, mu-metal-shielded vacuum components, minimising the volume of steel near the experiment and ensuring that high-current devices were shielded or far away.

Another of the interesting features of a CAEIS is the beam shaping capabilities which are especially interesting when considering space charge. Space charge degrades the brightness of charged particle beams but uniform ellipsoidal profile bunches preserve beam brightness. The beam shaping capabilities of a CAEIS allow the production of ellipsoidal bunches if the shaping is implemented with three dimensions of control, rather than the two-dimensional proof-of-concept implementation demonstrated so far with this apparatus. In Chapter 5 a new emittance measurement technique is described and demonstrated, using streaked pepperpots to determine the time-resolved brightness of a beam. The accuracy of the technique was demonstrated with comparison to theoretical predictions and simulations over two dramatically different timescales. This technique should prove useful when quantifying the efficacy of methods to reduce beam degradation due to space charge. While the implementation was somewhat

constrained by the geometry and current of the source the technique is generally applicable to charged particle beams and could prove useful in contexts other than cold-atom charged particle sources.

It had been hoped that the CAES could be used to demonstrate CDI with relatively simple samples but this was not feasible due to the combination of beam drift and the low beam current. If the beam trajectories had been stable then CDI could have been performed with long measurement times averaging many thousands of bunches. Significantly larger beam current could result in enough signal in a single shot that the registration algorithms would have been able to compensate for the drift and average out multiple shots.

The CAEIS used in this research has reached the end of its useful life. The apparatus has achieved a number of milestones along the road to molecular imaging: low-temperature electron and ion production [46, 47, 50, 76], arbitrary beam shaping [54], space-charge observation and manipulation [49, 62, 159], single-shot diffraction [1], and now ultrafast diffraction.

It still remains to be seen if electrons generated from the photoionisation of laser cooled atoms can be used to perform ultrafast, single-shot coherent diffractive imaging of arbitrary molecules, let alone if such a source would be competitive with its rivals such as X-ray free electron lasers (XFELs) or photocathode electron sources. Due to the high brightness of cold-atom sources they show promise as an ion source for use with ion microscopes and focused ion beam milling [63].

The lessons learned during the course of the research described by this thesis have informed the design of the next iteration of CAEIS at the University of Melbourne. The new apparatus will have the aim of creating a more reliable, higher-current source that takes advantage of a modern electron or ion microscope column. This thesis also explored the limits of frequency stabilisation with PS and it has been shown that PS is able to achieve sub-kilohertz frequency linewidth, previously only reachable with high-finesse optical cavities. A new technique for the measurement of the brightness of charged particle beams with time resolution has also been described and demonstrated, and should prove to be a useful tool for the exploration of space-charge dynamics in a CAEIS. The first stage of the next-generation cold-atom source has already been constructed and used to demonstrate electron-ion coincidence for producing individual heralded and gated single ions [79]. It is hoped that further development of the new cold-atom source, based on the knowledge gained from the studies described in this thesis and by fellow students and collaborators will make substantial steps towards achieving single-shot diffractive imaging of complex atomic-scale targets.

Bibliography

1. Speirs, R. W., Putkunz, C. T., McCulloch, A. J., Nugent, K. A., Sparkes, B. M. & Scholten, R. E. "Single-shot electron diffraction using a cold atom electron source". *Journal of Physics B: Atomic, Molecular and Optical Physics* **48,** 214002 (2015) (cited on pages iv, 6, 66, 70, 71, 75, 99, 105).
2. Watson, J. D. & Crick, F. H. C. "Molecular Structure of Nucleic Acids: A Structure for Deoxyribose Nucleic Acid". *Nature* **171,** 737–738 (Apr. 1953) (cited on page 1).
3. Kendrew, J. C. & Perutz, M. F. "X-Ray Studies of Compounds of Biological Interest". *Annual Review of Biochemistry* **26,** 327–372 (1957) (cited on page 1).
4. Dwyer, J. R., Hebeisen, C. T., Ernstorfer, R., Harb, M., Deyirmenjian, V. B., Jordan, R. E. & Dwayne Miller, R. J. "Femtosecond Electron Diffraction: Making the Molecular Movie". *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences* **364,** 741–778 (Mar. 2006) (cited on pages 1, 2).
5. Sciaini, G. & Dwayne Miller, R. J. "Femtosecond electron diffraction: heralding the era of atomically resolved dynamics". *Reports on Progress in Physics* **74,** 096101 (2011) (cited on pages 1, 2, 77).
6. McClelland, J. J., Steele, A. V., Knuffman, B., Twedt, K. A., Schwarzkopf, A. & Wilson, T. M. "Bright focused ion beam sources based on laser-cooled atoms". *Applied Physics Reviews* **3,** 011302 (Mar. 2016) (cited on page 1).
7. Knuffman, B., Steele, A. V. & McClelland, J. J. "Cold atomic beam ion source for focused ion beam applications". *Journal of Applied Physics* **114,** 044303 (July 2013) (cited on pages 1, 7).
8. van Oudheusden, T., de Jong, E. F. & van der Geer, S. B. "Electron source concept for single-shot sub-100 fs electron diffraction in the 100 keV range". *Journal of Applied Physics* **102,** 093501 (Nov. 2007) (cited on pages 1, 77).
9. Zhu, T. & Dürr, H. "The future of electron microscopy". *Physics Today* **68,** 32–38 (Mar. 2015) (cited on page 1).

10. McCulloch, A. J., Sparkes, B. M. & Scholten, R. E. “Cold electron sources using laser-cooled atoms”. *Journal of Physics B: Atomic, Molecular and Optical Physics* **49,** 164004 (2016) (cited on pages 1, 66).

11. Hardy, L. W., Finer-Moore, J. S., Montfort, W. R., Jones, M. O., Santi, D. V. & Stroud, R. M. “Atomic structure of thymidylate synthase: target for rational drug design”. *Science* **235,** 448–455 (Jan. 1987) (cited on page 1).

12. Barrett, M. D. & Workman, P. “Discovering novel chemotherapeutic drugs for the third millennium”. *European Journal of Cancer* **35,** 2010–2030 (Dec. 1999) (cited on page 1).

13. Pinto, L. H., Holsinger, L. J. & Lamb, R. A. “Influenza virus M2 protein has ion channel activity”. *Cell* **69,** 517–528 (May 1992) (cited on page 1).

14. Cullity, B. D. & Stock, S. R. *Elements of X-ray Diffraction* (Prentice Hall, 2001) (cited on page 2).

15. Bacon, G. E. *X-Ray and Neutron Diffraction* (Elsevier, Sept. 2013) (cited on page 2).

16. Dauter, Z. “Current state and prospects of macromolecular crystallography”. *Acta Crystallographica Section D: Biological Crystallography* **62,** 1–11 (Jan. 2006) (cited on page 2).

17. Levitt, M. “Nature of the protein universe”. *Proceedings of the National Academy of Sciences* **106,** 11079–11084 (July 2009) (cited on page 2).

18. Henderson, R., Baldwin, J. M., Ceska, T. A., Zemlin, F., Beckmann, E. & Downing, K. H. “Model for the structure of bacteriorhodopsin based on high-resolution electron cryo-microscopy”. *Journal of Molecular Biology* **213,** 899–929 (June 1990) (cited on page 2).

19. Zhou, Z. H. “Towards atomic resolution structural determination by single-particle cryo-electron microscopy”. *Current Opinion in Structural Biology* **18,** 218–228 (Apr. 2008) (cited on page 2).

20. Gaffney, K. J. & Chapman, H. N. “Imaging Atomic Structure and Dynamics with Ultrafast X-ray Scattering”. *Science* **316,** 1444–1448 (June 2007) (cited on pages 2, 3).

21. Barty, A., Boutet, S., Bogan, M. J., Hau-Riege, S., Marchesini, S., Sokolowski-Tinten, K., Stojanovic, N., Tobey, R., Ehrke, H., Cavalleri, A., Düsterer, S., Frank, M., Bajt, S., Woods, B. W., Seibert, M. M., Hajdu, J., Treusch, R. & Chapman, H. N. “Ultrafast single-shot diffraction imaging of nanoscale dynamics”. *Nature Photonics* **2,** 415–419 (July 2008) (cited on page 2).

22. Miao, J., Ishikawa, T., Robinson, I. K. & Murnane, M. M. “Beyond crystallography: Diffractive imaging using coherent x-ray light sources”. *Science* **348,** 530–535 (May 2015) (cited on page 2).

23. Chapman, H. N., Barty, A., Bogan, M. J., Boutet, S., Frank, M., Hau-Riege, S. P., Marchesini, S., Woods, B. W., Bajt, S., Benner, W. H., London, R. A., Plönjes, E., Kuhlmann, M., Treusch, R., Düsterer, S., Tschentscher, T., Schneider, J. R., Spiller, E., Möller, T., Bostedt, C., Hoener, M., Shapiro, D. A., Hodgson, K. O., van der Spoel, D., Burmeister, F., Bergh, M., Caleman, C., Huldt, G., Seibert, M. M., Maia, F. R. N. C., Lee, R. W., Szöke, A., Timneanu, N. & Hajdu, J. "Femtosecond diffractive imaging with a soft-X-ray free-electron laser". *Nature Physics* **2,** 839–843 (Dec. 2006) (cited on pages 2, 3).

24. Yefanov, O. M. & Vartanyants, I. A. "Orientation determination in single-particle x-ray coherent diffraction imaging experiments". *Journal of Physics B: Atomic, Molecular and Optical Physics* **46,** 164013 (2013) (cited on page 2).

25. Kupitz, C., Olmos, J. L., Holl, M., Tremblay, L., Pande, K., Pandey, S., Oberthür, D., Hunter, M., Liang, M., Aquila, A., Tenboer, J., Calvey, G., Katz, A., Chen, Y., Wiedorn, M. O., Knoska, J., Meents, A., Majriani, V., Norwood, T., Poudyal, I., Grant, T., Miller, M. D., Xu, W., Tolstikova, A., Morgan, A., Metz, M., Martin-Garcia, J. M., Zook, J. D., Roy-Chowdhury, S., Coe, J., Nagaratnam, N., Meza, D., Fromme, R., Basu, S., Frank, M., White, T., Barty, A., Bajt, S., Yefanov, O., Chapman, H. N., Zatsepin, N., Nelson, G., Weierstall, U., Spence, J., Schwander, P., Pollack, L., Fromme, P., Ourmazd, A., Phillips, G. N. & Schmidt, M. "Structural enzymology using X-ray free electron lasers". *Structural Dynamics* **4,** 044003 (Dec. 2016) (cited on page 2).

26. Pande, K., Hutchison, C. D. M., Groenhof, G., Aquila, A., Robinson, J. S., Tenboer, J., Basu, S., Boutet, S., DePonte, D. P., Liang, M., White, T. A., Zatsepin, N. A., Yefanov, O., Morozov, D., Oberthuer, D., Gati, C., Subramanian, G., James, D., Zhao, Y., Koralek, J., Brayshaw, J., Kupitz, C., Conrad, C., Roy-Chowdhury, S., Coe, J. D., Metz, M., Xavier, P. L., Grant, T. D., Koglin, J. E., Ketawala, G., Fromme, R., Šrajer, V., Henning, R., Spence, J. C. H., Ourmazd, A., Schwander, P., Weierstall, U., Frank, M., Fromme, P., Barty, A., Chapman, H. N., Moffat, K., van Thor, J. J. & Schmidt, M. "Femtosecond Structural Dynamics Drives the Trans/Cis Isomerization in Photoactive Yellow Protein". *Science (New York, N.Y.)* **352,** 725–729 (May 2016) (cited on page 2).

27. Nango, E., Royant, A., Kubo, M., Nakane, T., Wickstrand, C., Kimura, T., Tanaka, T., Tono, K., Song, C., Tanaka, R., Arima, T., Yamashita, A., Kobayashi, J., Hosaka, T., Mizohata, E., Nogly, P., Sugahara, M., Nam, D., Nomura, T., Shimamura, T., Im, D., Fujiwara, T., Yamanaka, Y., Jeon, B., Nishizawa, T., Oda, K., Fukuda, M., Andersson, R., Båth, P., Dods, R., Davidsson, J., Matsuoka, S., Kawatake, S., Murata, M., Nureki, O., Owada, S., Kameshima, T., Hatsui, T., Joti, Y., Schertler, G., Yabashi, M., Bondar, A.-N., Standfuss, J., Neutze, R. & Iwata, S. "A three-dimensional movie of structural changes in bacteriorhodopsin". *Science* **354,** 1552–1557 (Dec. 2016) (cited on page 2).

28. Huldt, G., Szoke, A. & Hajdu, J. “Diffraction imaging of single particles and biomolecules”. *Journal of Structural Biology* **144,** 219–227 (Nov. 2003) (cited on page 3).

29. Spence, J. C. H. “Outrunning damage: Electrons vs X-rays - timescales and mechanisms”. *Structural Dynamics* **4,** 044027 (June 2017) (cited on page 3).

30. Rodenburg, J. M. “The phase problem, microdiffraction and wavelength-limited resolution - a discussion”. *Ultramicroscopy* **27,** 413–422 (May 1989) (cited on page 3).

31. Chapman, H. N. & Nugent, K. A. “Coherent lensless X-ray imaging”. *Nature Photonics* **4,** 833–839 (Dec. 2010) (cited on page 3).

32. Putkunz, C. T., D’Alfonso, A. J., Morgan, A. J., Weyland, M., Dwyer, C., Bourgeois, L., Etheridge, J., Roberts, A., Scholten, R. E., Nugent, K. A. & Allen, L. J. “Atom-Scale Ptychographic Electron Diffractive Imaging of Boron Nitride Cones”. *Physical Review Letters* **108,** 073901 (Feb. 2012) (cited on pages 3, 77).

33. Aloy, P. & Russell, R. B. “Structural systems biology: modelling protein interactions”. *Nature Reviews Molecular Cell Biology* **7,** 188–197 (Mar. 2006) (cited on page 4).

34. Almèn, M. S., Nordström, K. J. V., Fredriksson, R. & Schiöth, H. B. “Mapping the human membrane proteome: a majority of the human membrane proteins can be classified according to function and evolutionary origin”. *BMC Biology* **7,** 50 (Aug. 2009) (cited on page 4).

35. Klebe, G. “Recent developments in structure-based drug design”. *Journal of Molecular Medicine (Berlin, Germany)* **78,** 269–281 (2000) (cited on page 4).

36. Jhoti, H. & Leach, A. R. *Structure-based drug discovery* (Springer, 2007) (cited on page 4).

37. Mauser, H. & Guba, W. “Recent developments in de novo design and scaffold hopping”. *Current Opinion in Drug Discovery & Development* **11,** 365–374 (May 2008) (cited on page 4).

38. Greer, J., Erickson, J. W., Baldwin, J. J. & Varney, M. D. “Application of the three-dimensional structures of protein target molecules in structure-based drug design”. *Journal of Medicinal Chemistry* **37,** 1035–1054 (Apr. 1994) (cited on page 4).

39. Geerlof, A., Brown, J., Coutard, B., Egloff, M. P., Enguita, F. J., Fogg, M. J., Gilbert, R. J. C., Groves, M. R., Haouz, A., Nettleship, J. E., Nordlund, P., Owens, R. J., Ruff, M., Sainsbury, S., Svergun, D. I. & Wilmanns, M. “The impact of protein characterization in structural proteomics”. *Acta Crystallographica. Section D, Biological Crystallography* **62,** 1125–1136 (Oct. 2006) (cited on page 4).

40. Engelen, W. J., Smakman, E. P., Bakker, D. J., Luiten, O. J. & Vredenbregt, E. J. D. “Effective temperature of an ultracold electron source based on near-threshold photoionization”. *Ultramicroscopy* **136,** 73–80 (Jan. 2014) (cited on page 5).

41. Baranov, L. Y., Kris, R., Levine, R. D. & Even, U. “On the field ionization spectrum of high Rydberg states”. *The Journal of Chemical Physics* **100,** 186–196 (Jan. 1994) (cited on page 5).

42. Dowell, D. H., Bethel, S. Z. & Friddell, K. D. “Results from the average power laser experiment photocathode injector test”. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment* **356,** 167–176 (Mar. 1995) (cited on page 5).

43. Engelen, W. J., van der Heijden, M. A., Bakker, D. J., Vredenbregt, E. J. D. & Luiten, O. J. “High-coherence electron bunches produced by femtosecond photoionization”. *Nature Communications* **4,** 1693 (Apr. 2013) (cited on page 5).

44. Engelen, W. J., Vredenbregt, E. J. D. & Luiten, O. J. “Analytical model of an isolated single-atom electron source”. *Ultramicroscopy* **147,** 61–69 (Dec. 2014) (cited on page 5).

45. Sparkes, B. M., Thompson, D. J., McCulloch, A. J., Murphy, D., Speirs, R. W., Torrance, J. S. J. & Scholten, R. E. “High-Coherence Electron and Ion Bunches From Laser-Cooled Atoms”. *Microscopy and Microanalysis* **20,** 1008–1014 (Aug. 2014) (cited on pages 5, 77).

46. Speirs, R. W., McCulloch, A. J., Sparkes, B. M. & Scholten, R. E. “Identification of competing ionization processes in the generation of ultrafast electron bunches from cold-atom electron sources”. *Physical Review A* **95,** 053408 (May 2017) (cited on pages 5, 6, 14, 16, 20, 66, 70, 73, 77, 102, 105).

47. Saliba, S. D., Putkunz, C. T., Sheludko, S. V., McCulloch, A. J., Nugent, K. A. & Scholten, R. E. “Spatial coherence of electron bunches extracted from an arbitrarily shaped cold atom electron source”. *Optics Express* **20,** 3967–3974 (Feb. 2012) (cited on pages 5, 6, 77, 82, 105).

48. Debernardi, N., Reijnders, M. P., Engelen, W. J., Clevis, T. T. J., Mutsaers, P. H. A., Luiten, O. J. & Vredenbregt, E. J. D. “Measurement of the temperature of an ultracold ion source using time-dependent electric fields”. *Journal of Applied Physics* **110,** 024501 (July 2011) (cited on page 5).

49. Murphy, D., Speirs, R. W., Sheludko, D. V., Putkunz, C. T., McCulloch, A. J., Sparkes, B. M. & Scholten, R. E. “Detailed observation of space-charge dynamics using ultracold ion bunches”. *Nature Communications* **5,** 4489 (July 2014) (cited on pages 5, 7, 105).

50. McCulloch, A. J., Speirs, R. W., Grimmel, J., Sparkes, B. M., Comparat, D. & Scholten, R. E. “Field ionization of Rydberg atoms for high-brightness electron and ion beams”. *Physical Review A* **95,** 063845 (June 2017) (cited on pages 6, 20, 81, 105).

51. Wieman, C. & Hänsch, T. W. “Doppler-Free Laser Polarization Spectroscopy”. *Physical Review Letters* **36,** 1170–1173 (May 1976) (cited on pages 6, 34, 36, 46, 47).

52. Torrance, J. S., Sparkes, B. M., Turner, L. D. & Scholten, R. E. “Sub-kilohertz laser linewidth narrowing using polarization spectroscopy”. *Optics Express* **24,** 11396–11406 (May 2016) (cited on pages 6, 8, 35, 37, 64).

53. van Oudheusden, T., Pasmans, P. L. E. M., van der Geer, S. B., de Loos, M. J., van der Wiel, M. J. & Luiten, O. J. “Compression of Subrelativistic Space-Charge-Dominated Electron Bunches for Single-Shot Femtosecond Electron Diffraction”. *Physical Review Letters* **105,** 264801 (Dec. 2010) (cited on pages 6, 20, 77).

54. McCulloch, A. J., Sheludko, D. V., Saliba, S. D., Bell, S. C., Junker, M., Nugent, K. A. & Scholten, R. E. “Arbitrarily shaped high-coherence electron bunches from cold atoms”. *Nature Physics* **7,** 785–788 (July 2011) (cited on pages 6, 16, 17, 89, 105).

55. Luiten, O. J., van der Geer, S. B., de Loos, M. J., Kiewiet, F. B. & van der Wiel, M. J. “How to Realize Uniform Three-Dimensional Ellipsoidal Electron Bunches”. *Physical Review Letters* **93,** 094802 (Aug. 2004) (cited on pages 6, 18, 77).

56. Thompson, D. J., Murphy, D., van Bijnen, R., Speirs, R. W., McCulloch, A. J., Scholten, R. E. & Sparkes, B. M. *Bunch Shaping for Improved Brightness with a Cold Atom Electron and Ion Source.* in (OSA, 2015), FW6B.1 (cited on page 6).

57. Torrance, J. S., Speirs, R. W., McCulloch, A. J. & Scholten, R. E. “Time-resolved brightness measurements by streaking”. *Physical Review Accelerators and Beams* **21,** 032802 (Mar. 2018) (cited on pages 6, 8, 76).

58. Scipioni, L., Stern, L. A., Notte, J., Sijbrandij, S. & Griffin, B. “Helium Ion Microscope”. *Advanced Materials & Processes* **166,** 27–30 (2008) (cited on pages 7, 77).

59. Khizroev, S. & Litvinov, D. “Focused-ion-beam-based rapid prototyping of nanoscale magnetic devices”. *Nanotechnology* **15,** R7 (2004) (cited on pages 7, 77).

60. Knuffman, B., Steele, A. V., Orloff, J. & McClelland, J. J. “Nanoscale focused ion beam from laser-cooled lithium atoms”. *New Journal of Physics* **13,** 103035 (2011) (cited on page 7).

61. Steele, A. V., Knuffman, B., McClelland, J. J. & Orloff, J. “Focused chromium ion beam”. *Journal of Vacuum Science & Technology B, Nanotechnology and Microelectronics: Materials, Processing, Measurement, and Phenomena* **28,** C6F1–C6F5 (Oct. 2010) (cited on page 7).

62. Thompson, D. J., Murphy, D., Speirs, R. W., van Bijnen, R. M. W., McCulloch, A. J., Scholten, R. E. & Sparkes, B. M. “Suppression of Emittance Growth Using a Shaped Cold Atom Electron and Ion Source”. *Physical Review Letters* **117,** 193202 (Nov. 2016) (cited on pages 7, 18, 77, 105).

63. Steele, A. V., Schwarzkopf, A., McClelland, J. J. & Knuffman, B. “High-brightness Cs focused ion beam from a cold-atomic-beam ion source”. *Nano Futures* **1,** 015005 (2017) (cited on pages 7, 105).

64. Sheludko, D. V. *Shaped Electron Bunches from Ultracold Plasma.* PhD Thesis (The University of Melbourne, Melbourne, Australia, Dec. 2010) (cited on page 9).

65. Bell, S. C. *Cold Electrons Extracted from an Ultracold Plasma.* PhD thesis (The University of Melbourne, School of Physics, 2011) (cited on pages 9, 11).

66. Saliba, S. D. *A cold atom electron source for diffractive imaging.* PhD thesis (University of Melbourne, School of Physics, 2011) (cited on pages 9, 37).

67. McCulloch, A. J. *Generation of shaped cold electron bunches for ultrafast electron diffraction.* PhD thesis (The University of Melbourne, Jan. 2013) (cited on pages 9, 13).

68. Taylor, R. J. *Rydberg Excitation of Rubidium to Produce an Ultra-Cold Plasma.* MA thesis (The University of Melbourne, School of Physics, Oct. 2013) (cited on page 9).

69. Tielen, R. *Development of a Continuous Cold Atom Electron & Ion Source.* MA thesis (The University of Melbourne, School of Physics, Oct. 2015) (cited on pages 9, 22).

70. Murphy, D. *Measurement and modelling of intrabeam coulomb interactions in ultracold ion bunches.* PhD thesis (The University of Melbourne, School of Physics, 2017) (cited on page 9).

71. Speirs, R. W. *Electron diffraction using a cold atom source.* PhD Thesis (2017) (cited on pages 9, 14, 15, 66, 70, 86, 100).

72. Bell, S. C., Junker, M., Jasperse, M., Turner, L. D., Lin, Y.-J., Spielman, I. B. & Scholten, R. E. “A slow atom source using a collimated effusive oven and a single-layer variable pitch coil Zeeman slower”. *The Review Of Scientific Instruments* **81,** 013105 (Jan. 2010) (cited on pages 11, 22).

73. Metcalf, H. J. & van der Straten, P. *Laser Cooling and Trapping* (Springer, 1999) (cited on pages 13, 34).

74. Hanssen, J. L., Dakin, E. A., McClelland, J. J. & Jacka, M. “Using laser-cooled atoms as a focused ion beam source”. *Journal of Vacuum Science & Technology B* **24,** 2907–2910 (2006) (cited on page 13).

75. Joachain, C. J., Kylstra, N. J. & Potvliege, R. M. *Atoms in Intense Laser Fields.* 2011 (cited on page 15).

76. McCulloch, A. J., Sheludko, D. V., Junker, M. & Scholten, R. E. “High-coherence picosecond electron bunches from cold atoms”. *Nature Communications* **4,** 1692 (Apr. 2013) (cited on pages 15, 77, 80, 99, 105).

77. Gallagher, T. F. *Rydberg Atoms* (Cambridge University Press, Nov. 2005) (cited on page 16).

78. van Bijnen, R. M. W., Ravensbergen, C., Bakker, D. J., Dijk, G. J., Kokkelmans, S. J. J. M. F. & Vredenbregt, E. J. D. "Patterned Rydberg excitation and ionization with a spatial light modulator". *New Journal of Physics* **17,** 023045 (2015) (cited on page 16).

79. McCulloch, A. J., Speirs, R. W., Wissenberg, S. H., Tielen, R. P. M., Sparkes, B. M. & Scholten, R. E. "Heralded ions via ionization coincidence". *Physical Review A* **97,** 043423 (Apr. 2018) (cited on pages 26, 105).

80. Fox, R. W., Oates, C. W. & Hollberg, L. W. in *Experimental Methods in the Physical Sciences* (eds van Zee, R. D. & Patrick Looney, J.) 1–46 (Academic Press, Jan. 2003) (cited on page 34).

81. Anderson, M. H., Ensher, J. R., Matthews, M. R., Wieman, C. E. & Cornell, E. A. "Observation of Bose-Einstein Condensation in a Dilute Atomic Vapor". *Science* **269,** 198–201 (July 1995) (cited on page 34).

82. DeMarco, B. & Jin, D. S. "Onset of Fermi Degeneracy in a Trapped Atomic Gas". *Science* **285,** 1703–1706 (Sept. 1999) (cited on page 34).

83. Uetake, S., Yamaguchi, A., Kato, S. & Takahashi, Y. "High power narrow linewidth laser at 556 nm for magneto-optical trapping of ytterbium". *Applied Physics B* **92,** 33–35 (July 2008) (cited on page 34).

84. Ye, L., Yi-Ge, L., Yang, Z., Qiang, W., Shao-Kai, W., Tao, Y., Jian-Ping, C., Tian-Chu, L., Zhan-Jun, F. & Er-Jun, Z. "Stable Narrow Linewidth 689 nm Diode Laser for the Second Stage Cooling and Trapping of Strontium Atoms". *Chinese Physics Letters* **27,** 074208 (July 2010) (cited on page 34).

85. Akamatsu, D., Nakajima, Y., Inaba, H., Hosaka, K., Yasuda, M., Onae, A. & Hong, F.-L. "Narrow linewidth laser system realized by linewidth transfer using a fiber-based frequency comb for the magneto-optical trapping of strontium". *Optics Express* **20,** 16010 (July 2012) (cited on page 34).

86. Ludlow, A. D., Zelevinsky, T., Campbell, G. K., Blatt, S., Boyd, M. M., Miranda, M. H. G. d., Martin, M. J., Thomsen, J. W., Foreman, S. M., Ye, J., Fortier, T. M., Stalnaker, J. E., Diddams, S. A., Coq, Y. L., Barber, Z. W., Poli, N., Lemke, N. D., Beck, K. M. & Oates, C. W. "Sr Lattice Clock at 1x10-16 Fractional Uncertainty by Remote Optical Evaluation with a Ca Clock". *Science* **319,** 1805–1808 (Mar. 2008) (cited on page 34).

87. Rafac, R. J., Young, B. C., Beall, J. A., Itano, W. M., Wineland, D. J. & Bergquist, J. C. "Sub-dekahertz Ultraviolet Spectroscopy of 199Hg+". *Physical Review Letters* **85,** 2462–2465 (Sept. 2000) (cited on page 34).

88. Ye, J., Kimble, H. J. & Katori, H. “Quantum State Engineering and Precision Metrology Using State-Insensitive Light Traps”. *Science* **320,** 1734–1738 (June 2008) (cited on page 34).

89. Haroche, S. & Hartmann, F. “Theory of Saturated-Absorption Line Shapes”. *Physical Review A* **6,** 1280–1300 (Oct. 1972) (cited on pages 34, 36).

90. Preston, D. W. “Doppler-free saturated absorption: Laser spectroscopy”. *American Journal of Physics* **64,** 1432 (1996) (cited on pages 34, 36).

91. Maguire, L. P., van Bijnen, R. M. W., Mese, E. & Scholten, R. E. “Theoretical calculation of saturated absorption spectra for multi-level atoms”. *Journal of Physics B: Atomic, Molecular and Optical Physics* **39,** 2709–2720 (June 2006) (cited on pages 34, 36).

92. Drever, R., Hall, J. L., Kowalski, F., Hough, J., Ford, G., Munley, A. & Ward, H. “Laser phase and frequency stabilization using an optical resonator”. *Applied Physics B* **31,** 97–105 (1983) (cited on pages 34, 36, 40).

93. Ludlow, A. D., Huang, X., Notcutt, M., Zanon-Willette, T., Foreman, S. M., Boyd, M. M., Blatt, S. & Ye, J. “Compact, thermal-noise-limited optical cavity for diode laser stabilization at 1x10 $^{-15}$”. *Optics Letters* **32,** 641 (2007) (cited on pages 34, 37).

94. Kessler, T., Hagemann, C., Grebing, C., Legero, T., Sterr, U., Riehle, F., Martin, M. J., Chen, L. & Ye, J. “A sub-40-mHz-linewidth laser based on a silicon single-crystal optical cavity”. *Nature Photonics* **6,** 687–692 (Oct. 2012) (cited on pages 34, 40).

95. Abramovici, A., Althouse, W. E., Drever, R. W. P., Gürsel, Y., Kawamura, S., Raab, F. J., Shoemaker, D., Sievers, L., Spero, R. E., Thorne, K. S., Vogt, R. E., Weiss, R., Whitcomb, S. E. & Zucker, M. E. “LIGO: The Laser Interferometer Gravitational-Wave Observatory”. *Science* **256,** 325–333 (Apr. 1992) (cited on page 34).

96. Black, E. D. “An introduction to Pound-Drever-Hall laser frequency stabilization”. *American Journal of Physics* **69,** 79–87 (Jan. 2001) (cited on pages 34, 40).

97. Demtröder, W. *Laser Spectroscopy: Basic Concepts and Instrumentation* (Springer Science & Business Media, 2003) (cited on pages 34, 39, 46).

98. Torii, Y., Tashiro, H., Ohtsubo, N. & Aoki, T. “Laser-phase and frequency stabilization using atomic coherence”. *Physical Review A* **86,** 033805 (Sept. 2012) (cited on pages 34, 36, 54, 56).

99. Yoshikawa, Y., Umeki, T., Mukae, T., Torii, Y. & Kuga, T. “Frequency Stabilization of a Laser Diode with Use of Light-Induced Birefringence in an Atomic Vapor”. *Applied Optics* **42,** 6645 (2003) (cited on pages 35, 36, 46, 63).

100. Systems, M. *ORS-DL Optical Reference System - ultra-narrow lasers in the visible range*. Dec. 2016 (cited on page 35).

101. Cuneo, C. J., Maki, J. J. & McIntyre, D. H. “Optically stabilized diode laser using high-contrast saturated absorption”. *Applied Physics Letters* **64,** 2625–2627 (May 1994) (cited on page 36).

102. Saliba, S. D. & Scholten, R. E. “Linewidths below 100 kHz with external cavity diode lasers”. *Applied Optics* **48,** 6961 (Dec. 2009) (cited on pages 36, 40).

103. Corwin, K. L., Lu, Z.-T., Hand, C. F., Epstein, R. J. & Wieman, C. E. “Frequency-Stabilized Diode Laser with the Zeeman Shift in an Atomic Vapor”. *Applied Optics* **37,** 3295–3298 (May 1998) (cited on page 36).

104. Millett-Sikking, A., Hughes, I. G., Tierney, P. & Cornish, S. L. “DAVLL lineshapes in atomic rubidium”. *Journal of Physics B: Atomic, Molecular and Optical Physics* **40,** 187 (Jan. 2007) (cited on page 36).

105. Shirley, J. H. “Modulation transfer processes in optical heterodyne saturation spectroscopy”. *Optics Letters* **7,** 537 (Nov. 1982) (cited on page 36).

106. McCarron, D. J., King, S. A. & Cornish, S. L. “Modulation transfer spectroscopy in atomic rubidium”. *Measurement Science and Technology* **19.** arXiv: 0805.2708, 105601 (Oct. 2008) (cited on page 36).

107. Xiang-Hui, Q., Wen-Lan, C., Lin, Y., Da-Wei, Z., Tong, Z., Qin, X., Jun, D., Xiao-Ji, Z. & Xu-Zong, C. “Ultra-Stable Rubidium-Stabilized External-Cavity Diode Laser Based on the Modulation Transfer Spectroscopy Technique”. *Chinese Physics Letters* **26,** 044205 (Apr. 2009) (cited on page 36).

108. Negnevitsky, V. & Turner, L. D. “Wideband laser locking to an atomic reference with modulation transfer spectroscopy”. *Optics Express* **21,** 3103 (Feb. 2013) (cited on page 36).

109. Robins, N. P., Slagmolen, B. J. J., Shaddock, D. A., Close, J. D. & Gray, M. B. “Interferometric, modulation-free laser stabilization”. *Optics Letters* **27,** 1905 (Nov. 2002) (cited on page 36).

110. Jundt, G., Purves, G. T., Adams, C. S. & Hughes, I. G. “Non-linear Sagnac interferometry for pump-probe dispersion spectroscopy”. *The European Physical Journal D - Atomic, Molecular, Optical and Plasma Physics* **27,** 273–276 (Dec. 2003) (cited on page 36).

111. Lancaster, G., Conroy, R., Clifford, M., Arlt, J. & Dholakia, K. “A polarisation spectrometer locked diode laser for trapping cold atoms”. *Optics Communications* **170,** 79–84 (Oct. 1999) (cited on page 36).

112. Harris, M. L., Adams, C. S., Cornish, S. L., McLeod, I. C., Tarleton, E. & Hughes, I. G. “Polarization spectroscopy in rubidium and cesium”. *Physical Review A* **73,** 062509 (June 2006) (cited on page 36).

113. Pearman, C. P., Adams, C. S., Cox, S. G., Griffin, P. F., Smith, D. A. & Hughes, I. G. "Polarization spectroscopy of a closed atomic transition: applications to laser frequency locking". *Journal of Physics B: Atomic, Molecular and Optical Physics* **35,** 5141 (Dec. 2002) (cited on pages 36, 45, 46).

114. Tiwari, V. B., Singh, S., Mishra, S. R., Rawat, H. S. & Mehendale, S. C. "Laser frequency stabilization using Doppler-free bi-polarization spectroscopy". *Optics Communications* **263,** 249–255 (July 2006) (cited on pages 36, 63).

115. Do, H. D., Moon, G. & Noh, H.-R. "Polarization spectroscopy of rubidium atoms: Theory and experiment". *Physical Review A* **77,** 032513 (Mar. 2008) (cited on page 36).

116. Hänsch, T. W. & Couillaud, B. "Laser frequency stabilization by polarization spectroscopy of a reflecting reference cavity". *Optics Communications* **35,** 441–444 (Dec. 1980) (cited on page 36).

117. Wieman, C. E. & Hollberg, L. "Using diode lasers for atomic physics". *Review of Scientific Instruments* **62,** 1–20 (Jan. 1991) (cited on pages 36, 60).

118. Hawthorn, C. J., Weber, K. P. & Scholten, R. E. "Littrow configuration tunable external cavity diode laser with fixed direction output beam". *Review of Scientific Instruments* **72,** 4477 (Dec. 2001) (cited on page 37).

119. Steck, D. A. *Rubidium 85 D Line Data.* Tech. rep. (Oregon Center for Optics and Department of Physics, University of Oregon, 2008) (cited on pages 40, 80).

120. Hecht, E. & Zajac, A. *Optics* (Addison-Wesley, 1987) (cited on page 41).

121. Pedrotti, F. L., Pedrotti, L. M. & Pedrotti, L. S. *Introduction to Optics* (Pearson Prentice Hall, 2007) (cited on page 41).

122. Hughes, I. G. "Polarization Spectroscopy of Alkali-Metal Atoms". *New Trends in Quantum Coherence and Nonlinear Optics* **263,** 149–169 (2009) (cited on pages 45, 46).

123. Himsworth, M. & Freegarde, T. "Rubidium pump-probe spectroscopy: Comparison between ab initio theory and experiment". *Physical Review A* **81,** 023423 (Feb. 2010) (cited on page 48).

124. Cooper, C. *Physics.* en (Taylor & Francis, 2001) (cited on page 52).

125. Okoshi, T., Kikuchi, K. & Nakayama, A. "Novel method for high resolution measurement of laser output spectrum". *Electronics Letters* **16,** 630–631 (July 1980) (cited on page 54).

126. Richter, L., Mandelberg, H., Kruger, M. & McGrath, P. "Linewidth determination from self-heterodyne measurements with subcoherence delay times". *IEEE Journal of Quantum Electronics* **22,** 2070–2074 (Nov. 1986) (cited on page 54).

127. Turner, L., Weber, K., Hawthorn, C. & Scholten, R. “Frequency noise characterisation of narrow linewidth diode lasers”. *Optics Communications* **201,** 391–397 (Jan. 2002) (cited on pages 57, 62).

128. Lee, M. W., Jarratt, M. C., Marciniak, C. & Biercuk, M. J. “Frequency Stabilization of a 369 nm Diode Laser by Nonlinear Spectroscopy of Ytterbium Ions in a Discharge”. *Optics Express* **22.** arXiv: 1402.1248, 7210 (Mar. 2014) (cited on page 63).

129. Thompson, D. J. & Scholten, R. E. “Narrow linewidth tunable external cavity diode laser using wide bandwidth filter”. *Review of Scientific Instruments* **83,** 023107 (Feb. 2012) (cited on page 64).

130. van der Geer, S. B., Loos, M. J. d., Vredenbregt, E. J. D. & Luiten, O. J. “Ultracold Electron Source for Single-Shot, Ultrafast Electron Diffraction”. *Microscopy and Microanalysis* **15,** 282–289 (Aug. 2009) (cited on page 66).

131. Bragg, W. H. & Bragg, W. L. “The structure of the diamond”. *Proc. R. Soc. Lond. A* **89,** 277–291 (Sept. 1913) (cited on page 66).

132. Zeitler, E. “Cryo electron microscopy”. *Ultramicroscopy* **10,** 1–5 (Jan. 1982) (cited on page 66).

133. Cowley, J. M. *Electron Diffraction Techniques* (International Union of Crystallography, 1992) (cited on page 66).

134. Kittel, C. *Introduction to solid state physics* 8th (Wiley, 2004) (cited on page 67).

135. Peng, L. M. “Electron atomic scattering factors and scattering potentials of crystals”. *Micron* **30,** 625–648 (Dec. 1999) (cited on page 68).

136. Reiser, M. *Theory and Design of Charged Particle Beams* (John Wiley & Sons, Aug. 2008) (cited on pages 76, 80).

137. Mills, A. P. “Brightness enhancement of slow positron beams”. *Applied physics* **23,** 189–191 (Oct. 1980) (cited on page 76).

138. Qiang, J., Lidia, S., Ryne, R. D. & Limborg-Deprey, C. “Three-dimensional quasistatic model for high brightness beam dynamics simulation”. *Physical Review Special Topics - Accelerators and Beams* **9,** 044204 (Apr. 2006) (cited on page 76).

139. Bazarov, I. V., Dunham, B. M. & Sinclair, C. K. “Maximum Achievable Beam Brightness from Photoinjectors”. *Physical Review Letters* **102,** 104801 (Mar. 2009) (cited on page 76).

140. Collins, L. E. & Stroud, P. T. “Extraction of high current ion beams with low divergence”. *Nuclear Instruments and Methods* **26,** 157–166 (Feb. 1964) (cited on page 76).

141. Van Steenbergen, A. “Evaluation of particle beam phase space measurement techniques”. *Nuclear Instruments and Methods* **51,** 245–253 (May 1967) (cited on page 76).

142. Wang, J. G., Wang, D. X. & Reiser, M. “Beam emittance measurement by the pepper-pot method”. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment* **307,** 190–194 (Oct. 1991) (cited on page 76).

143. Brunetti, E., Shanks, R. P., Manahan, G. G., Islam, M. R., Ersfeld, B., Anania, M. P., Cipiccia, S., Issac, R. C., Raj, G., Vieux, G., Welsh, G. H., Wiggins, S. M. & Jaroszynski, D. A. “Low Emittance, High Brilliance Relativistic Electron Beams from a Laser-Plasma Accelerator”. *Physical Review Letters* **105,** 215007 (Nov. 2010) (cited on page 76).

144. Zhang, M. *Emittance Formula for Slits and Pepper-pot Measurement.* Oct. 1996 (cited on pages 76, 81, 93, 97).

145. Delerue, N., Bartolini, R., Peach, K., Reichold, A., Senanayake, R., Bajlekov, S., Caballero-Bendixsen, L., Ibbotson, T., Thomas, C., Nicolas Bourgeois, Buonomo, B., Doucas, G., Hooker, S., Lau, P., Urner, D. & Walker, P. A. *Single-Shot Emittance Measurement of a 508MeV Electron Beam Using the Pepper-Pot Method.* in *Particle accelerator. Proceedings, 23rd Conference, PAC'09, Vancouver, Canada, May 4-8, 2009* (2010), TH5RFP065 (cited on page 76).

146. Collier, J., Hall, G., Haseroth, H., Kugler, H., Kuttenberger, A., Langbein, K., Scrivens, R., Sherwood, T., Tambini, J., Sharkov, B., Shumshurov, A. & Masek, K. “The CERN laser-ion source”. *Laser and Particle Beams* **14,** 283–292 (Sept. 1996) (cited on page 76).

147. Yoshida, M., Hasegawa, J., Fukata, S., Oguri, Y., Ogawa, M., Nakajima, M., Horioka, K., Maebara, S. & Shiho, M. “A simple time-resolved emittance measurement of a laser ion source with a digital camera”. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment. Pfoc. of the 13th Int. Symp. on Heavy Ion Intertial Fusion* **464,** 582–586 (May 2001) (cited on page 76).

148. Walter, M., Lamb, D., Bernal, S., Haber, I., Kishek, R. A., Li, H., Quinn, B., Snowel, M., Valfells, A., Reiser, M. & O'Shea, P. O. *Time resolved emittance measurement in the University of Maryland Electron Ring.* in *Proceedings of the 2003 Particle Accelerator Conference* **4** (May 2003), 2574–2576 vol.4 (cited on page 76).

149. Li, R., Huang, W., Du, Y., Yan, L., Du, Q., Shi, J., Hua, J., Chen, H., Du, T., Xu, H. & Tang, C. “Note: Single-shot continuously time-resolved MeV ultrafast electron diffraction”. *Review of Scientific Instruments* **81,** 036110 (Mar. 2010) (cited on pages 77, 82, 102).

150. Mukojima, K., Kanzaki, S., Kawanishi, K., Sato, K. & Abukawa, T. “Streak-camera reflection high-energy electron diffraction for dynamics of surface crystallography”. *Surface Science* **636,** 25–30 (June 2015) (cited on page 77).

151. Ratner, D., Behrens, C., Ding, Y., Huang, Z., Marinelli, A., Maxwell, T. & Zhou, F. "Time-resolved imaging of the microbunching instability and energy spread at the Linac Coherent Light Source". *Physical Review Special Topics - Accelerators and Beams* **18,** 030704 (Mar. 2015) (cited on page 77).

152. Siwick, B. J., Dwyer, J. R., Jordan, R. E. & Dwayne Miller, R. J. "An Atomic-Level View of Melting Using Femtosecond Electron Diffraction". *Science* **302,** 1382–1385 (Nov. 2003) (cited on page 77).

153. Tokita, S., Inoue, S., Masuno, S., Hashida, M. & Sakabe, S. "Single-shot ultrafast electron diffraction with a laser-accelerated sub-MeV electron pulse". *Applied Physics Letters* **95,** 111911 (Sept. 2009) (cited on page 77).

154. Flewett, S., Quiney, H. M., Tran, C. Q. & Nugent, K. A. "Extracting coherent modes from partially coherent wavefields". *Optics Letters* **34,** 2198–2200 (July 2009) (cited on page 77).

155. Abbey, B., Whitehead, L. W., Quiney, H. M., Vine, D. J., Cadenazzi, G. A., Henderson, C. A., Nugent, K. A., Balaur, E., Putkunz, C. T., Peele, A. G., Williams, G. J. & McNulty, I. "Lensless imaging using broadband X-ray sources". *Nature Photonics* **5,** 420–424 (July 2011) (cited on page 77).

156. Kassier, G. H., Haupt, K., Erasmus, N., Rohwer, E. G., von Bergmann, H. M., Schwoerer, H., Coelho, S. M. M. & Auret, F. D. "A compact streak camera for 150 fs time resolved measurement of bright pulses in ultrafast electron diffraction". *Review of Scientific Instruments* **81,** 105103 (Oct. 2010) (cited on pages 82, 102).

157. van Rens, J. F. M., Verhoeven, W., Franssen, J. G. H., Lassise, A. C., Stragier, X. F. D., Kieft, E. R., Mutsaers, P. H. A. & Luiten, O. J. "Theory and particle tracking simulations of a resonant radiofrequency deflection cavity in TM110 mode for ultrafast electron microscopy". *Ultramicroscopy* **184,** 77–89 (Jan. 2018) (cited on pages 82, 102).

158. Musumeci, P., Moody, J. T., Scoby, C. M., Gutierrez, M. S., Bender, H. A. & Wilcox, N. S. "High quality single shot diffraction patterns using ultrashort megaelectron volt electron beams from a radio frequency photoinjector". *Review of Scientific Instruments* **81,** 013306 (Jan. 2010) (cited on page 102).

159. Murphy, D., Scholten, R. E. & Sparkes, B. M. "Increasing the Brightness of Cold Ion Beams by Suppressing Disorder-Induced Heating with Rydberg Blockade". *Physical Review Letters* **115,** 214802 (Nov. 2015) (cited on page 105).

Appendix A

Glossary

AOM acousto-optic modulator.

AR anti-reflection.

BS beam splitter.

CAEIS cold-atom electron and ion source.

CAES cold-atom electron source.

CAIS cold-atom ion source.

CCD charge-coupled device.

CDI coherent diffractive imaging.

CW continuous wave.

DAVLL dichroic atomic vapour laser lock.

ECDL external cavity diode laser.

EOM electro-optic modulator.

FIB focused ion beam.

FSR free-spectral range.

FWHM full-width half maximum.

LIGO the Laser Interferometer Gravitational-Wave Observatory.

MCP microchannel plate.

MOT magneto-optical trap.

MTS modulation transfer spectroscopy.

PBS polarising beam splitter.

PDH Pound-Drever-Hall.

PS polarisation spectroscopy.

PSD power spectral density.

PSF point spread function.

REMPE resonace-enhanced multiphoton excitation.

RF radio frequency.

RMS root mean square.

SA saturated absorption spectroscopy.

SLM spatial light modulator.

SNR signal-to-noise ratio.

TA tapered amplifier.

TCMPE two-colour multiphoton excitation.

TEC thermo-electric cooler.

TEM transmission electron miscroscopy.

UED ultrafast electron diffraction.

XFEL X-ray free electron laser.

Appendix B

Code

The code presented in this appendix was written for use with Python 3.

B.1 One-Dimensional Pepperpot Simulation

This code was used to simulate the behaviour of the electron beam as the emittance is measured with a pepperpot mask. The results of the simulations performed with this code confirmed the accuracy of the corrections applied to the emittance measurements. Bunch1d.py contains the core simulator code, Mask1D.py contains functions for applying pepperpot masks to the beam, and SimScript1D.py contains simulations of a number of different scenarios. These simulations are discussed in Section 5.4.

B.1.1 Bunch1D.py

```
##### Imports #####
from numpy import array, zeros, sqrt
from numpy.random import randn
from scipy.constants import m_e, c
from matplotlib import pyplot as plt

##### Object #####
class Bunch:
    def __init__(self, n=1, energy=1, rms_width=1, \
                 rms_emittance=None, normalised_rms_emittance=None,\
                 mass=m_e, blank=False):
        if not blank:
            self.setMass(mass)
            self.setEnergy(energy)

            self._createBunch(n=n, rms_width=rms_width, \
                              rms_emittance=rms_emittance, \
                              normalised_rms_emittance=normalised_rms_emittance)

    def _createBunch(self, n=1, rms_width=1, rms_emittance=None, \
                     normalised_rms_emittance=None):
        self.electrons = randn(n, 2)
            # x, vx

        self.electrons[:, 0] = self.electrons[:, 0]*rms_width

        if normalised_rms_emittance!=None:
            self.setNormalisedRMSEmittance(normalised_rms_emittance)
```

```
    elif rms_emittance!=None:
        self.setRMSEmittance(rms_emittance)
    else:
        self.electrons[:, 1] = zeros(n)

def copy(self):
    new_bunch = Bunch(blank=True)

    new_bunch.electrons = self.electrons.copy()

    new_bunch.speed = self.speed

    return new_bunch

def __iter__(self):
    return self.electrons.__iter__()

def next(self):
    return self.electrons.next()

def propagate(self, dz):
    dt = dz / self.getSpeed()

    self.electrons[:, 0] += self.electrons[:, 1]*dt

def getBeta(self):
    return self.getSpeed() / c

def getRMSEmittance(self):
    xs = self.getXs()
    xprimes = self.getXPrimes()

    exs = xs-xs.mean()
    exs_p = xprimes-xprimes.mean()

    return sqrt((exs**2).mean() * (exs_p**2).mean() - (exs*exs_p).mean()**2)

def setRMSEmittance(self, rms_emittance):
    if rms_emittance==0:
        self.electrons[:, 1] = zeros(self.electrons.shape[0])
    else:
        self.electrons[:, 1] = randn(self.electrons.shape[0])

        rms_position = self.getXs().std()

        rms_divergence = rms_emittance / rms_position

        self.electrons[:, 1] *= rms_divergence * self.getSpeed()

def getNormalisedRMSEmittance(self):
    return self.getBeta() * self.getRMSEmittance()

def setNormalisedRMSEmittance(self, norm_rms_emittance):
    self.setRMSEmittance(norm_rms_emittance/self.getBeta())

def getSize(self):
    return self.electrons.shape[0]

def getWidth(self):
    return self.getXs().std()

def getMass(self):
    return self.mass

def setMass(self, mass):
    self.mass = mass

def getEnergy(self):
    return self.getMass()*self.getSpeed()**2 / 2

def setEnergy(self, energy):
    self.setSpeed(sqrt(2*energy/self.mass))
```

```
    def getSpeed(self):
        return self.speed

    def setSpeed(self, speed):
        self.speed = speed

    def getXs(self):
        return self.electrons[:, 0]

    def setXs(self, Xs):
        self.electrons[:, 0] = Xs

    def getVXs(self):
        return self.electrons[:, 1]

    def setVXs(self, VXs):
        self.electrons[:, 1] = VXs

    def getXPrimes(self):
        return self.getVXs() / self.getSpeed()

    def plot_phasespace(self, figname=None, color=None):
        plt.figure(figname)

        # Plot X and Px
        plt.title('X')
        plt.xlabel('X')
        plt.ylabel('Px')

        plt.plot(self.getXs(), self.getVXs()*m_e, ',', c=color)
```

B.1.2 Mask1D.py

```
##### Imports #####
from numpy import array

from Bunch1D import Bunch


##### Functions #####
def maskBunch(bunch, mask_function):
    bunch.electrons = bunch.electrons[list(map(mask_function, bunch.getXs()))]

    return bunch

def pinholeMask(pinhole_diameter=50e-6, location=0):
    pinhole_function = lambda x: abs(x-location) < pinhole_diameter/2

    return pinhole_function

def pepperpotMaskFs(pinhole_diameter=50e-6, pitch=200e-6, number_holes=7, location=0):
    fs = []
    for i in range(number_holes):
        x = (i - number_holes/2 + 0.5)*pitch + location

        f = pinholeMask(pinhole_diameter, location=x)

        fs.append(f)

    return fs

def pepperpotMask(pinhole_diameter=50e-6, pitch=200e-6, number_holes=7, location=0):
    fs = pepperpotMaskFs(pinhole_diameter=pinhole_diameter, pitch=pitch, \
                         number_holes=number_holes, location=location)

    func = lambda x: any(list(map(lambda f: f(x), fs)))

    return func

def maskBunch2(bunch, pepperpotFs):
    bunch.electrons = bunch.electrons[list(map(lambda x: any(map(lambda f: f(x), \
                                                pepperpotFs)), bunch.getXs()))]
```

```
    return bunch
```

B.1.3 SimScript1D.py

```
##### Imports #####
from numpy import histogram, sqrt, arange, array, linspace, isnan, zeros, flipud, logspace, isnan
from matplotlib import rcParams, pyplot as plt
from peakutils import indexes, interpolate
from scipy.constants import eV, m_e, k as kB, c
from scipy.ndimage.filters import gaussian_filter
from scipy.stats import norm
from h5py import File

from Bunch1D import Bunch
from Mask1D import maskBunch, maskBunch2, pinholeMask, pepperpotMask, pepperpotMaskFs
from Emittance import emittance_from_line, excess_energy_from_wavelength, expected_emittance, normal_prop
from Fitting import fitCurve

##### Constants #####
beam_energy = 8e3*eV
beam_size_rms = 570.68e-6
beam_size_rms_cherry = 340e-6

##### Functions #####
def find_peaks(x, line, thres=0.001, min_dist=200, smooth=False, smooth_size=5, diag=False):
    if smooth:
        l = gaussian_filter(line, smooth_size)
    else:
        l = line

    peaks = indexes(l, thres=thres, min_dist=min_dist)

    if diag:
        plt.figure('Peaks')
        plt.title('Peak␣Finding')
        plt.plot(line, 'b')
        if smooth:
            plt.plot(l, 'r')
        for peak in peaks:
            plt.axvline(peak, color='k', ls=':')

    return peaks

def simple_lens(bunch, strength=1):
    bunch.electrons[:, 1] += bunch.getXs() * strength

def spherical_aberration(bunch, strength=1):
    bunch.electrons[:, 1] += bunch.getXs()**2 * strength

def normal_cumulative_thingy(x):
    return 1-(1-norm.cdf(x))*2

def normal_prop(x_lo, x_hi):
    return norm.cdf(x_hi) - norm.cdf(x_lo)


##### Script #####
# Excess energy of steaks
if False:
    long_streak_wavelength = 487.2e-9
    short_streak_wavelength = 475.9e-9

    long_streak_excess = excess_energy_from_wavelength(long_streak_wavelength, field_ionisation=False)
    short_streak_excess = excess_energy_from_wavelength(short_streak_wavelength, field_ionisation=False)

    print('Long␣duration␣streak:')
    print('\tWavelength:␣{:.2f}nm'.format(long_streak_wavelength*1e9))
    print('\tExcess␣Energy:␣{:.2f}meV'.format(long_streak_excess*1e3))

    print()

    print('Short␣duration␣streak:')
    print('\tWavelength:␣{:.2f}nm'.format(short_streak_wavelength*1e9))
```

```
    print('\tExcess Energy: {:.2f}meV'.format(short_streak_excess*1e3))

    # For thesis figure from arbitrary shaping paper.
    wavelengths = [481.729, 481.185, 479.920, 479.152, \
                   483.300, 482.200, 481.987, 481.670, \
                   480.570, 479.865, 478.799, 477.658, 476.694, 475.438]

    print('\nFigure Energies:')
    for wav in wavelengths:
        e = excess_energy_from_wavelength(wav*1e-9, field_ionisation=True, electric_field=40e3)

        print('Wavelength: {:.3f}nm, Excess Energy: {:.4f}meV'.format(wav, e*1e3))

    exit()

# First Run.
if False:
    num_particles = int(1e6)

    expected_temperature = 5
    expected_emittance = beam_size_rms * sqrt(kB*expected_temperature/(m_e*c**2))

    # Pepperpots
    number_of_holes = 20
    pepperpot_pitch = 200e-6

    propagation_distance = 0.2


    print('Expected Emittance: {:.2f} nm rad'.format(expected_emittance*1e9))
    bunch = Bunch(n=num_particles, energy=beam_energy, rms_width=beam_size_rms, \
                  normalised_rms_emittance=expected_emittance, mass=m_e)
    print('Initial emittance: {:.2f} nm rad'.format(bunch.getNormalisedRMSEmittance()*1e9))

    hist, bin_edges = histogram(bunch.getXs(), bins=10000)
    plt.figure('hist')
    plt.plot(bin_edges[:-1], hist)

    print('Pre-Pepperpot emittance: {:.2f} nm rad'.format(bunch.getNormalisedRMSEmittance()*1e9))
    pinhole = pepperpotMask(pitch=pepperpot_pitch, number_holes=number_of_holes)
    bunch = maskBunch(bunch, pinhole)
    print('Post-Pepperpot emittance:', bunch.getNormalisedRMSEmittance()*1e9)

    hist, bin_edges = histogram(bunch.getXs(), bins=bin_edges)
    plt.figure('hist')
    plt.plot(bin_edges[:-1], hist)

    dz = propagation_distance
    bunch.propagate(dz)
    print('Final emittance:', bunch.getNormalisedRMSEmittance()*1e9)

    hist, bin_edges = histogram(bunch.getXs(), bins=bin_edges)
    plt.figure('hist')
    plt.plot(bin_edges[:-1], hist)

    m_per_pixel = bin_edges[1]-bin_edges[0]
    peaks = find_peaks(bin_edges[:-1], hist, diag=False)
    hist = array(hist, dtype='f')
    plt.figure()
    emittance, rms_size, total_count = emittance_from_line(hist, peaks, sum_peaks=True, diag=True,
                        m_per_pixel=m_per_pixel, number_holes=number_of_holes,
                        pitch=pepperpot_pitch, propagation_distance=propagation_distance)
    emittance *= bunch.getBeta()

    print('Measured Emittance: {:.2f} nm rad'.format(emittance*1e9))

# Wavelength Sweep
if False:
    if False:
        beam_size = beam_size_rms
        filename = 'wavelength_sweep - long2.h5'
    else:
        beam_size = beam_size_rms_cherry
```

```
    filename = 'wavelength_sweep - long2 min.h5'

if False:
    # Simulate and save
    wavelengths = linspace(465e-9, 487e-9, 100)

    excess_energys = excess_energy_from_wavelength(wavelengths, field_ionisation=False)

    expected_emittances = expected_emittance(excess_energys, beam_size)

    # Pepperpot
    number_of_holes = 15
    pepperpot_pitch = 200e-6
    aperture_size = 50e-6
    pepperpot = pepperpotMask(pitch=pepperpot_pitch, number_holes=number_of_holes, pinhole_diameter=aperture_size)

    # Bins for histogram 'detector'
    if False:
        bins = 1000
    else:
        bins = linspace(-.0017, 0.0017, 1000)

    N = 10000000
    propagation_distance = 0.015
    # For each +'ve excess energy simulate a bunch.
    measured_emittances = zeros(wavelengths.size)
    measured_emittances_beam_size_corrected = zeros(wavelengths.size)
    measured_emittances_aperture_size_corrected = zeros(wavelengths.size)
    for i, expected_e in enumerate(expected_emittances):
        if isnan(expected_e):
            if False:
                # Skip
                measured_emittances[i] = float('nan')
                measured_emittances_aperture_size_corrected[i] = float('nan')
                continue
            else:
                # Use zero emittance beam.
                expected_e = 0
                expected_emittances[i] = expected_e

        print(i)
        print('Emittance: {:.2f}nm rad'.format(1e9*expected_e))

        bunch = Bunch(n=N, energy=beam_energy, rms_width=beam_size, \
                      normalised_rms_emittance=expected_e, mass=m_e)

        print('\tBunch RMS Size: {:.2f}um'.format(bunch.getWidth()*1e6))

        bunch = maskBunch(bunch, pepperpot)

        bunch.propagate(propagation_distance)

        hist, bin_edges = histogram(bunch.getXs(), bins=bins)
        m_per_pixel = bin_edges[1]-bin_edges[0]
        hist = array(hist, dtype='f')

        peaks = find_peaks(bin_edges[:-1], hist, thres=0.005, min_dist=50, diag=False, smooth=True)


        plt.figure()

        # Don't correct for anything.
        emittance, rms_size, total_count = emittance_from_line(hist, peaks, sum_peaks=True, diag=True,
                        m_per_pixel=m_per_pixel, number_holes=len(peaks),
                        pitch=pepperpot_pitch, propagation_distance=propagation_distance,
                        adjust_for_size=False, adjust_for_aperture_size=False, refine_parameters=True)
        emittance *= bunch.getBeta()

        measured_emittances[i] = emittance

        # Correct for beam size.
        emittance, rms_size, total_count = emittance_from_line(hist, peaks, sum_peaks=True, diag=False,
                        m_per_pixel=m_per_pixel, number_holes=len(peaks),
```

```
                    pitch=pepperpot_pitch, propagation_distance=propagation_distance,
                    adjust_for_size=True, adjust_for_aperture_size=False, refine_parameters=False)
        emittance *= bunch.getBeta()

        measured_emittances_beam_size_corrected[i] = emittance

        # Now with aperture size correction.
        emittance, _rms_size, _total_count = emittance_from_line(hist, peaks, sum_peaks=True, diag=False,
                    m_per_pixel=m_per_pixel, number_holes=len(peaks),
                    pitch=pepperpot_pitch, propagation_distance=propagation_distance,
                    adjust_for_size=True, adjust_for_aperture_size=True, refine_parameters=True)
        emittance *= bunch.getBeta()
        measured_emittances_aperture_size_corrected[i] = emittance

        print('\tMeasured Emittance: {:.2f}nm rad'.format(emittance*1e9))

        if False:
            plt.figure('hist')
            plt.plot(bin_edges[:-1], hist)

            plt.show()
            exit()


    # Save data
    if True:
        with File(filename) as hdf:
            wavelengths_key = 'wavelengths'
            if wavelengths_key in hdf:
                del hdf[wavelengths_key]

            hdf.create_dataset(wavelengths_key, data=wavelengths)

            excess_energy_key = 'excess energy'
            if excess_energy_key in hdf:
                del hdf[excess_energy_key]

            hdf.create_dataset(excess_energy_key, data=excess_energys)

            expected_emittance_key = 'expected emittance'
            if expected_emittance_key in hdf:
                del hdf[expected_emittance_key]

            hdf.create_dataset(expected_emittance_key, data=expected_emittances)

            measured_emittance_key = 'measured emittance'
            if measured_emittance_key in hdf:
                del hdf[measured_emittance_key]

            hdf.create_dataset(measured_emittance_key, data=measured_emittances)

            measured_emittance_key = 'measured emittance beam size correction'
            if measured_emittance_key in hdf:
                del hdf[measured_emittance_key]

            hdf.create_dataset(measured_emittance_key, data=measured_emittances_beam_size_corrected)

            measured_emittance_key = 'measured emittance aperture size correction'
            if measured_emittance_key in hdf:
                del hdf[measured_emittance_key]

            hdf.create_dataset(measured_emittance_key, data=measured_emittances_aperture_size_corrected)

            hdf.attrs['number of holes'] = number_of_holes
            hdf.attrs['aperture size'] = aperture_size
            hdf.attrs['pitch'] = pepperpot_pitch
else:
    with File(filename) as hdf:
        wavelengths = array(hdf['wavelengths'])
        excess_energys = array(hdf['excess energy'])
        expected_emittances = array(hdf['expected emittance'])
        measured_emittances = array(hdf['measured emittance'])
        measured_emittances_beam_size_corrected = array(hdf['measured emittance beam size correction'])
```

```
            measured_emittances_aperture_size_corrected = array(hdf['measured␣emittance␣aperture␣size␣correction'])

            number_of_holes = hdf.attrs['number␣of␣holes']
            aperture_size = hdf.attrs['aperture␣size']
            pepperpot_pitch = hdf.attrs['pitch']


    # Plotting
    plt.figure('Wavelength␣vs.␣Excess␣Energy')
    plt.title('Wavelength␣vs.␣Excess␣Energy')
    plt.plot(wavelengths*1e9, excess_energys*1e3)
    plt.xlabel('Wavelength␣(nm)')
    plt.ylabel('Excess␣Energy␣(meV)')
    plt.axhline(0, color='k', ls=':')
    plt.xlim((wavelengths.min()*1e9, wavelengths.max()*1e9))


    plt.figure('Excess␣Emittance␣vs.␣Emittance')
    plt.title('Excess␣Emittance␣vs.␣Emittance')
    plt.plot(excess_energys*1e3, expected_emittances*1e9, 'r')
    plt.plot(excess_energys*1e3, measured_emittances*1e9, 'bx')
    plt.plot(excess_energys*1e3, measured_emittances_beam_size_corrected*1e9, 'gx')
    plt.plot(excess_energys*1e3, measured_emittances_aperture_size_corrected*1e9, 'kx')
    plt.xlabel('Excess␣Energy␣(meV)')
    plt.ylabel('Emittance␣(nm␣rad)')
    plt.xlim((0, excess_energys.max()*1e3))

    if True:
        # Plot for thesis.
        rcParams.update({'font.size': 10})
        rcParams.update({'pgf.rcfonts': False})
        rcParams.update({'pgf.texsystem': 'pdflatex'})

        colours = [(79/255,122/255,174/255),(255/255,102/255,51/255),(245/255,174/255,32/255),(77/255,155/255,77/255),(102/255,102/255,102

        # Font should match document
        rcParams['font.family'] = 'serif'

        rcParams['axes.unicode_minus'] = False
            # Minus sign from matplot lib is too long for my taste.


        linewidth = 5.71 # inches

        figwidth = linewidth
        figheight = figwidth/3
        figwidth *= 0.75
        figsize = (figwidth, figheight)

        # Higher res theory
        wavelengths_fine = linspace(465e-9, 487e-9, 10000)

        excess_energys_fine = excess_energy_from_wavelength(wavelengths_fine, field_ionisation=False)

        expected_emittances_fine = expected_emittance(excess_energys_fine, beam_size)
        for i in range(expected_emittances_fine.size):
            if isnan(expected_emittances_fine[i]):
                expected_emittances_fine[i] = 0

        # Plot
        plt.figure(figsize=figsize)
        plt.plot(excess_energys*1e3, measured_emittances*1e9, '-', markersize=5, color=colours[0])
        plt.plot(excess_energys*1e3, measured_emittances_aperture_size_corrected*1e9, '-', markersize=5, color=colours[3])
        plt.plot(excess_energys_fine*1e3, expected_emittances_fine*1e9, ':', color=colours[1])
        plt.xlim((-5, excess_energys.max()*1e3))
        plt.xlabel('Excess␣Energy␣(meV)')
        plt.ylabel('Emittance␣(nm␣rad)')

        plt.tight_layout()

        plt.savefig('wavelength_sweep_sim.pgf')

# Attempting to show resolution limit.
```

```
if False:
    file_name = 'resolution_limit_demo.h5'

    N = 1000000
    d_source_to_lens = .25+.450
    d_lens_to_sample = .100
    d_sample_to_detector = .45+.430

    lens_strength = -1.0e9

    pepperpot_pitch = 200e-6
    number_of_holes = 21

    if False:
        # Simulate and Save.
        wavelengths = linspace(465e-9, 479e-9, 100)

        excess_energys = excess_energy_from_wavelength(wavelengths, field_ionisation=False)

        expected_emittances = expected_emittance(excess_energys, beam_size_rms)

        mn = -0.025
        mx = 0.025
        rn = mx - mn
        bins = linspace(mn - rn, mx + rn, 1000)


        aperture_sizes = array([10e-6, 50e-6, 100e-6])

        measured_emittances = zeros((expected_emittances.size, aperture_sizes.size))
        measured_corrected_emittances = zeros((expected_emittances.size, aperture_sizes.size))
        measured_blurred_emittances = zeros((expected_emittances.size, aperture_sizes.size))
        measured_beam_size = zeros((expected_emittances.size, aperture_sizes.size))
        for j, expected_emittance in enumerate(expected_emittances):
            print('Emittance:␣{:.2f}nm␣rad'.format(1e9*expected_emittance))
            base_bunch = Bunch(n=N, energy=beam_energy, rms_width=beam_size_rms, \
                        normalised_rms_emittance=expected_emittance, mass=m_e)
            print('Initial␣bunch␣emittance:␣{:.2f}nm␣rad'.format(base_bunch.getNormalisedRMSEmittance()*1e9))

            base_bunch.propagate(d_source_to_lens)
            simple_lens(base_bunch, strength=lens_strength)
            print('After␣lens␣bunch␣emittance:␣{:.2f}nm␣rad'.format(base_bunch.getNormalisedRMSEmittance()*1e9))
            base_bunch.propagate(d_lens_to_sample)
            print('Beam␣size␣at␣sample:␣{:.2f}mm'.format(base_bunch.getWidth()*1e3))
            rms_beam_size_sample = base_bunch.getWidth()


            print()

            for k, aperture_size in enumerate(aperture_sizes):

                print('Pepperpot␣aperture␣size:␣{:.2f}um'.format(aperture_size*1e6))
                # Pepperpot
                pepperpot = pepperpotMask(pitch=pepperpot_pitch, number_holes=number_of_holes, \
                                          location=0, pinhole_diameter=aperture_size)

                bunch = base_bunch.copy()

                bunch = maskBunch(bunch, pepperpot)
                print('\tAfter␣sample␣bunch␣emittance:␣{:.2f}nm␣rad'.format(bunch.getNormalisedRMSEmittance()*1e9))
                print('\tAfter␣sample␣bunch␣count:␣{:d}'.format(bunch.getSize()))

                bunch.propagate(d_sample_to_detector)

                hist, bin_edges = histogram(bunch.getXs(), bins=bins)
                m_per_pixel = bin_edges[1]-bin_edges[0]

                hist = array(hist, dtype='f')

                #plt.figure()
                peaks = find_peaks(bin_edges[:-1], hist, min_dist=25, diag=False, smooth=True)
```

```
        #plt.figure()
        emittance, rms_size, total_count = emittance_from_line(hist, peaks, sum_peaks=True,
                        diag=False, m_per_pixel=m_per_pixel, number_holes=len(peaks),
                        pitch=pepperpot_pitch, propagation_distance=d_sample_to_detector,
                        adjust_for_size=True, adjust_for_aperture_size=False, refine_parameters=True)
        emittance *= bunch.getBeta()

        corrected_emittance, rms_size, total_count = emittance_from_line(hist, peaks, sum_peaks=True, \
                        diag=False, m_per_pixel=m_per_pixel, number_holes=len(peaks),
                        pitch=pepperpot_pitch, propagation_distance=d_sample_to_detector,
                        aperture_size=aperture_size, adjust_for_size=True, \
                        adjust_for_aperture_size=True, refine_parameters=True)
        corrected_emittance *= bunch.getBeta()

        if True:
            # Blur the detected data to replicate the mcp psf
            psf_width_pixels = 35e-6 / m_per_pixel
            hist = gaussian_filter(hist, psf_width_pixels)

            blurred_emittance, rms_size, total_count = emittance_from_line(hist, peaks, \
                        sum_peaks=True, diag=False, m_per_pixel=m_per_pixel, \
                        number_holes=len(peaks), pitch=pepperpot_pitch, \
                        propagation_distance=d_sample_to_detector,
                        aperture_size=aperture_size, adjust_for_size=True, \
                        adjust_for_aperture_size=True, refine_parameters=True)
            blurred_emittance *= bunch.getBeta()

            measured_blurred_emittances[j, k] = blurred_emittance

        measured_emittances[j, k] = emittance
        measured_corrected_emittances[j, k] = corrected_emittance
        measured_beam_size[j, k] = rms_size

        print('\tMeasured␣Emittance:␣{:.2f}nm␣rad'.format(emittance*1e9))
        print('\tMeasured␣Emittance:␣{:.2f}nm␣rad'.format(corrected_emittance*1e9))
        print('\tMeasured␣Blurred␣Emittance:␣{:.2f}nm␣rad'.format(blurred_emittance*1e9))


# Save data
with File(file_name) as hdf:
    aperture_size_key = 'aperture␣size'
    if aperture_size_key in hdf:
        del hdf[aperture_size_key]

    hdf.create_dataset(aperture_size_key, data=aperture_sizes)

    expected_emittance_key = 'expected␣emittance'
    if expected_emittance_key in hdf:
        del hdf[expected_emittance_key]

    hdf.create_dataset(expected_emittance_key, data=expected_emittances)

    excess_energy_key = 'excess␣energy'
    if excess_energy_key in hdf:
        del hdf[excess_energy_key]

    hdf.create_dataset(excess_energy_key, data=excess_energys)

    measured_emittance_key = 'measured␣emittance'
    if measured_emittance_key in hdf:
        del hdf[measured_emittance_key]

    hdf.create_dataset(measured_emittance_key, data=measured_emittances)


    measured_corrected_emittance_key = 'measured␣corrected␣emittance'
    if measured_corrected_emittance_key in hdf:
        del hdf[measured_corrected_emittance_key]

    hdf.create_dataset(measured_corrected_emittance_key, data=measured_corrected_emittances)

    measured__blurred_corrected_emittance_key = 'measured␣blurred␣corrected␣emittance'
    if measured__blurred_corrected_emittance_key in hdf:
```

```
            del hdf[measured__blurred_corrected_emittance_key]

        hdf.create_dataset(measured__blurred_corrected_emittance_key, data=measured_blurred_emittances)

        measured_beam_size_key = 'measured beam size'
        if measured_beam_size_key in hdf:
            del hdf[measured_beam_size_key]

        hdf.create_dataset(measured_beam_size_key, data=measured_beam_size)

        hdf.attrs['rms beam width at pepperpot'] = rms_beam_size_sample
else:
    # Load data
    with File(file_name) as hdf:
        rms_beam_size_sample = hdf.attrs['rms beam width at pepperpot']
        excess_energys = array(hdf['excess energy'])
        expected_emittances = array(hdf['expected emittance'])
        aperture_sizes = array(hdf['aperture size'])
        measured_emittances = array(hdf['measured emittance'])
        measured_corrected_emittances = array(hdf['measured corrected emittance'])
        measured_blurred_emittances = array(hdf['measured blurred corrected emittance'])

if True:
    # Plot Stuff
    for i in range(3):
        plt.figure()

        plt.title('Aperture Size: {:.2f}um'.format(aperture_sizes[i]*1e6))
        plt.plot(excess_energys*1e3, expected_emittances*1e9, 'r:')
        plt.plot(excess_energys*1e3, measured_emittances[:, i]*1e9)
        plt.plot(excess_energys*1e3, measured_corrected_emittances[:, i]*1e9)
        plt.plot(excess_energys*1e3, measured_blurred_emittances[:, i]*1e9)
        plt.xlabel('Excess Energy (meV)')
        plt.ylabel('Measured Emittance (nm rad)')




    if False:
        # Plot for thesis.
        rcParams.update({'font.size': 10})
        rcParams.update({'pgf.rcfonts': False})
        rcParams.update({'pgf.texsystem': 'pdflatex'})

        # Font should match document
        rcParams['font.family'] = 'serif'

        rcParams['axes.unicode_minus'] = False
            # Minus sign from matplot lib is too long for my taste.


        linewidth = 5.71 # inches

        figwidth = 0.95*linewidth
        figheight = figwidth/3*1.44
        figsize = (figwidth, figheight)

        plt.figure(figsize=figsize)
        plt.plot(aperture_sizes*1e6, measured_emittances*1e9)
        plt.plot(aperture_sizes*1e6, measured_corrected_emittances*1e9)
        plt.xlabel('Aperture Size ($\mu$m)')
        plt.ylabel('Measured Emittance (nm rad)')
        plt.axhline(expected_e*1e9, ls=':', color='k')

        plt.tight_layout()

        plt.savefig('resolution_limit_sim.pgf')

# Attempting to show resolution limit - taking larger MCP psf into account.
if False:
    file_name = 'resolution_limit_psf_demo.h5'

    N = 1000000
```

```
d_source_to_lens = .25+.450
d_lens_to_sample = .100
d_sample_to_detector = .45+.430

lens_strength = -1.0e9

pepperpot_pitch = 200e-6
number_of_holes = 21

if False:
    # Simulate and Save.
    #wavelengths = linspace(465e-9, 479e-9, 100)
    wavelengths = linspace(475e-9, 479e-9, 50)

    excess_energys = excess_energy_from_wavelength(wavelengths, field_ionisation=False)

    expected_emittances = expected_emittance(excess_energys, beam_size_rms)

    mn = -0.025
    mx = 0.025
    rn = mx - mn
    bins = linspace(mn - rn, mx + rn, 1000)


    aperture_sizes = array([10e-6, 50e-6, 100e-6])
    aperture_sizes = array([50e-6])

    measured_emittances = zeros((expected_emittances.size, aperture_sizes.size))
    measured_corrected_emittances = zeros((expected_emittances.size, aperture_sizes.size))
    measured_blurred_emittances = zeros((expected_emittances.size, aperture_sizes.size))
    measured_beam_size = zeros((expected_emittances.size, aperture_sizes.size))
    for j, expected_emittance in enumerate(expected_emittances):
        print('Emittance: {:.2f}nm rad'.format(1e9*expected_emittance))
        base_bunch = Bunch(n=N, energy=beam_energy, rms_width=beam_size_rms, \
                      normalised_rms_emittance=expected_emittance, mass=m_e)
        print('Initial bunch emittance: {:.2f}nm rad'.format(base_bunch.getNormalisedRMSEmittance()*1e9))

        base_bunch.propagate(d_source_to_lens)
        simple_lens(base_bunch, strength=lens_strength)
        print('After lens bunch emittance: {:.2f}nm rad'.format(base_bunch.getNormalisedRMSEmittance()*1e9))
        base_bunch.propagate(d_lens_to_sample)
        print('Beam size at sample: {:.2f}mm'.format(base_bunch.getWidth()*1e3))
        rms_beam_size_sample = base_bunch.getWidth()


        print()

        for k, aperture_size in enumerate(aperture_sizes):

            print('Pepperpot aperture size: {:.2f}um'.format(aperture_size*1e6))
            # Pepperpot
            pepperpot = pepperpotMask(pitch=pepperpot_pitch, number_holes=number_of_holes, \
                                      location=0, pinhole_diameter=aperture_size)

            bunch = base_bunch.copy()

            bunch = maskBunch(bunch, pepperpot)
            print('\tAfter sample bunch emittance: {:.2f}nm rad'.format(bunch.getNormalisedRMSEmittance()*1e9))
            print('\tAfter sample bunch count: {:d}'.format(bunch.getSize()))

            bunch.propagate(d_sample_to_detector)

            hist, bin_edges = histogram(bunch.getXs(), bins=bins)
            m_per_pixel = bin_edges[1]-bin_edges[0]

            hist = array(hist, dtype='f')

            #plt.figure()
            peaks = find_peaks(bin_edges[:-1], hist, min_dist=25, diag=False, smooth=True)


            #plt.figure()
            emittance, rms_size, total_count = emittance_from_line(hist, peaks, sum_peaks=True,
```

```
                    diag=False, m_per_pixel=m_per_pixel, number_holes=len(peaks),
                    pitch=pepperpot_pitch, propagation_distance=d_sample_to_detector,
                    adjust_for_size=True, adjust_for_aperture_size=False, refine_parameters=True)
        emittance *= bunch.getBeta()

        corrected_emittance, rms_size, total_count = emittance_from_line(hist, peaks, sum_peaks=True, \
                    diag=False, m_per_pixel=m_per_pixel, number_holes=len(peaks),
                    pitch=pepperpot_pitch, propagation_distance=d_sample_to_detector,
                    aperture_size=aperture_size, adjust_for_size=True, \
                    adjust_for_aperture_size=True, refine_parameters=True)
        corrected_emittance *= bunch.getBeta()

        if True:
            # Blur the detected data to replicate the mcp psf
            psf_width_pixels = 4*35e-6 / m_per_pixel
            hist = gaussian_filter(hist, psf_width_pixels)

            blurred_emittance, rms_size, total_count = emittance_from_line(hist, peaks, \
                    sum_peaks=True, diag=False, m_per_pixel=m_per_pixel, \
                    number_holes=len(peaks), pitch=pepperpot_pitch, \
                    propagation_distance=d_sample_to_detector,
                    aperture_size=aperture_size, adjust_for_size=True, \
                    adjust_for_aperture_size=True, refine_parameters=True)
            blurred_emittance *= bunch.getBeta()

            measured_blurred_emittances[j, k] = blurred_emittance

        measured_emittances[j, k] = emittance
        measured_corrected_emittances[j, k] = corrected_emittance
        measured_beam_size[j, k] = rms_size

        print('\tMeasured␣Emittance:␣{:.2f}nm␣rad'.format(emittance*1e9))
        print('\tMeasured␣Emittance:␣{:.2f}nm␣rad'.format(corrected_emittance*1e9))
        print('\tMeasured␣Blurred␣Emittance:␣{:.2f}nm␣rad'.format(blurred_emittance*1e9))


# Save data
with File(file_name) as hdf:
    aperture_size_key = 'aperture␣size'
    if aperture_size_key in hdf:
        del hdf[aperture_size_key]

    hdf.create_dataset(aperture_size_key, data=aperture_sizes)

    expected_emittance_key = 'expected␣emittance'
    if expected_emittance_key in hdf:
        del hdf[expected_emittance_key]

    hdf.create_dataset(expected_emittance_key, data=expected_emittances)

    excess_energy_key = 'excess␣energy'
    if excess_energy_key in hdf:
        del hdf[excess_energy_key]

    hdf.create_dataset(excess_energy_key, data=excess_energys)

    measured_emittance_key = 'measured␣emittance'
    if measured_emittance_key in hdf:
        del hdf[measured_emittance_key]

    hdf.create_dataset(measured_emittance_key, data=measured_emittances)


    measured_corrected_emittance_key = 'measured␣corrected␣emittance'
    if measured_corrected_emittance_key in hdf:
        del hdf[measured_corrected_emittance_key]

    hdf.create_dataset(measured_corrected_emittance_key, data=measured_corrected_emittances)

    measured__blurred_corrected_emittance_key = 'measured␣blurred␣corrected␣emittance'
    if measured__blurred_corrected_emittance_key in hdf:
        del hdf[measured__blurred_corrected_emittance_key]
```

```
            hdf.create_dataset(measured__blurred_corrected_emittance_key, data=measured_blurred_emittances)

            measured_beam_size_key = 'measured␣beam␣size'
            if measured_beam_size_key in hdf:
                del hdf[measured_beam_size_key]

            hdf.create_dataset(measured_beam_size_key, data=measured_beam_size)

            hdf.attrs['rms␣beam␣width␣at␣pepperpot'] = rms_beam_size_sample
    else:
        # Load data
        with File(file_name) as hdf:
            rms_beam_size_sample = hdf.attrs['rms␣beam␣width␣at␣pepperpot']
            excess_energys = array(hdf['excess␣energy'])
            expected_emittances = array(hdf['expected␣emittance'])
            aperture_sizes = array(hdf['aperture␣size'])
            measured_emittances = array(hdf['measured␣emittance'])
            measured_corrected_emittances = array(hdf['measured␣corrected␣emittance'])
            measured_blurred_emittances = array(hdf['measured␣blurred␣corrected␣emittance'])

    if True:
        # Plot Stuff
        for i in range(aperture_sizes.size):
            plt.figure()

            plt.title('Aperture␣Size:␣{:.2f}um'.format(aperture_sizes[i]*1e6))
            plt.plot(excess_energys*1e3, expected_emittances*1e9, 'r:')
            plt.plot(excess_energys*1e3, measured_emittances[:, i]*1e9)
            plt.plot(excess_energys*1e3, measured_corrected_emittances[:, i]*1e9)
            plt.plot(excess_energys*1e3, measured_blurred_emittances[:, i]*1e9)
            plt.xlabel('Excess␣Energy␣(meV)')
            plt.ylabel('Measured␣Emittance␣(nm␣rad)')




        if False:
            # Plot for thesis.
            rcParams.update({'font.size': 10})
            rcParams.update({'pgf.rcfonts': False})
            rcParams.update({'pgf.texsystem': 'pdflatex'})

            # Font should match document
            rcParams['font.family'] = 'serif'

            rcParams['axes.unicode_minus'] = False
                # Minus sign from matplot lib is too long for my taste.


            linewidth = 5.71 # inches

            figwidth = 0.95*linewidth
            figheight = figwidth/3*1.44
            figsize = (figwidth, figheight)

            plt.figure(figsize=figsize)
            plt.plot(aperture_sizes*1e6, measured_emittances*1e9)
            plt.plot(aperture_sizes*1e6, measured_corrected_emittances*1e9)
            plt.xlabel('Aperture␣Size␣($\mu$m)')
            plt.ylabel('Measured␣Emittance␣(nm␣rad)')
            plt.axhline(expected_e*1e9, ls=':', color='k')

            plt.tight_layout()

            plt.savefig('resolution_limit_psf_sim.pgf')


# Realistic Parameters - Looking at beam coverage by pepperpots and centring.
if False:
    N = 1000000
    d_source_to_lens = .25+.450
    d_lens_to_sample = .100
    d_sample_to_detector = .45+.430
```

```
lens_strength = -1.0e9

pepperpot_pitch = 300e-6

if False:
    # Simulate and Save.
    wavelengths = array([475e-9])

    excess_energys = excess_energy_from_wavelength(wavelengths, field_ionisation=False)

    expected_emittances = expected_emittance(excess_energys, beam_size_rms)

    expected_e = expected_emittances[0]

    # To save time, propagate a bunch to the pepperpots and reuse.
    print('Emittance: {:.2f}nm rad'.format(1e9*expected_e))
    base_bunch = Bunch(n=N, energy=beam_energy, rms_width=beam_size_rms, \
                normalised_rms_emittance=expected_e, mass=m_e)
    print('Initial bunch emittance: {:.2f}nm rad'.format(base_bunch.getNormalisedRMSEmittance()*1e9))

    base_bunch.propagate(d_source_to_lens)
    simple_lens(base_bunch, strength=lens_strength)
    print('After lens bunch emittance: {:.2f}nm rad'.format(base_bunch.getNormalisedRMSEmittance()*1e9))
    base_bunch.propagate(d_lens_to_sample)
    print('Beam size at sample: {:.2f}mm'.format(base_bunch.getWidth()*1e3))
    rms_beam_size_sample = base_bunch.getWidth()

    mn = -0.025
    mx = 0.025
    rn = mx - mn
    bins = linspace(mn - rn, mx + rn, 1000)


    centres = linspace(0, 4*rms_beam_size_sample, 20)
    num_holess = array([25])#flipud(arange(2, 40, 1))
    measured_emittances = zeros((centres.size, num_holess.size))
    measured_beam_size = zeros((centres.size, num_holess.size))
    for k, pepperpot_center in enumerate(centres):
        for j, num_holes in enumerate(num_holess):
            # Pepperpot
            number_of_holes = num_holes
            pepperpot = pepperpotMask(pitch=pepperpot_pitch, number_holes=number_of_holes, \
                                      location=pepperpot_center)

            print('Pepperpot Centre: {:.2f}um'.format(pepperpot_center*1e6))
            print('Number of Holes:', num_holes)

            bunch = base_bunch.copy()

            bunch = maskBunch(bunch, pepperpot)
            print('\tAfter sample bunch emittance: {:.2f}nm rad'.format(bunch.getNormalisedRMSEmittance()*1e9))

            bunch.propagate(d_sample_to_detector)

            hist, bin_edges = histogram(bunch.getXs(), bins=bins)
            m_per_pixel = bin_edges[1]-bin_edges[0]

            peaks = find_peaks(bin_edges[:-1], hist, min_dist=25, diag=False, smooth=True)

            hist = array(hist, dtype='f')
            plt.figure()
            emittance, rms_size, total_count = emittance_from_line(hist, peaks, \
                            sum_peaks=True, diag=False,
                            m_per_pixel=m_per_pixel, number_holes=len(peaks),
                            pitch=pepperpot_pitch, propagation_distance=d_sample_to_detector, \
                            adjust_for_aperture_size=True, refine_parameters=True, adjust_for_size=False)
            emittance *= bunch.getBeta()

            measured_emittances[k, j] = emittance
            measured_beam_size[k, j]

            print('\tMeasured Emittance: {:.2f}nm rad'.format(emittance*1e9))
```

```
    # Save data
    with File('pepperpot_extent.h5') as hdf:
        number_of_holes_key = 'number of holes'
        if number_of_holes_key in hdf:
            del hdf[number_of_holes_key]

        hdf.create_dataset(number_of_holes_key, data=num_holess)

        centres_key = 'pepperpot centre'
        if centres_key in hdf:
            del hdf[centres_key]

        hdf.create_dataset(centres_key, data=centres)

        measured_emittance_key = 'measured emittance'
        if measured_emittance_key in hdf:
            del hdf[measured_emittance_key]

        hdf.create_dataset(measured_emittance_key, data=measured_emittances)

        measured_beam_size_key = 'measured beam size'
        if measured_beam_size_key in hdf:
            del hdf[measured_beam_size_key]

        hdf.create_dataset(measured_beam_size_key, data=measured_beam_size)

        hdf.attrs['expected emittance'] = expected_e
        hdf.attrs['rms beam width at pepperpot'] = rms_beam_size_sample
else:
    # Load data
    with File('pepperpot_extent.h5') as hdf:
        expected_e = hdf.attrs['expected emittance']
        rms_beam_size_sample = hdf.attrs['rms beam width at pepperpot']
        num_holess = array(hdf['number of holes'])
        measured_emittances = array(hdf['measured emittance'])
        centres = array(hdf['pepperpot centre'])

if False:
    # Plotting
    xs = num_holess*pepperpot_pitch/rms_beam_size_sample
    exs = linspace(xs.min(), xs.max(), 1000)
    wys = normal_cumulative_thingy(exs/2.66)
    correction = normal_cumulative_thingy(xs/2.66)

    fit_func = lambda x, k: normal_cumulative_thingy(x/k)
    guess = [3]
    fit = fitCurve(xs, measured_emittances[0, :]/expected_e, fit_func, guess)
    print(fit)

    # Different number of holes
    plt.figure('Number of holes vs. Measured Emittance')
    plt.plot(num_holess, measured_emittances[0, :]*1e9, label='Measured')
    plt.plot(num_holess, measured_emittances[0, :]*1e9/correction, label='Corrected')
    plt.axhline(expected_e*1e9, ls=':', color='r')
    plt.xlabel('Number of Pepperpot Holes')
    plt.ylabel('Emittance (nm rad)')
    plt.legend(loc='lower right')


    plt.figure('Normalised Number of Holes vs. Measured E')
    plt.plot(xs, measured_emittances[0, :]/expected_e, label='Measured')
    plt.plot(exs, wys, label='Correction')
    plt.axhline(1, color='k', ls=':')
    plt.xlabel('Pepperpot Extent/Beam Size')
    plt.ylabel('Proportion of Expected Emittance')
    plt.legend(loc='lower right')

    # Off Centre 'pots
    corrections = zeros(measured_emittances.shape)
    for i, c in enumerate(centres):
        for j, n in enumerate(num_holess):
```

```
            x_lo = (c - n*pepperpot_pitch/2.66) / rms_beam_size_sample
            x_hi = (c + n*pepperpot_pitch/2.66) / rms_beam_size_sample

            corrections[i, j] = 1/normal_prop(x_lo, x_hi)

    plt.figure('Off␣Centre␣Pepperpot␣-␣Attempted␣Correction')
    plt.title('Off␣Centre␣Pepperpot␣-␣Attempted␣Correction')
    corrected_emittance = corrections * measured_emittances
    plt.plot(centres*1e3, 1e9*measured_emittances[:, 0], 'b:')
    plt.plot(centres*1e3, 1e9*corrected_emittance[:, 0], 'b', label='{:d}␣holes'.format(num_holess[0]))
    plt.plot(centres*1e3, 1e9*measured_emittances[:, 10], 'g:')
    plt.plot(centres*1e3, 1e9*corrected_emittance[:, 10], 'g', label='{:d}␣holes'.format(num_holess[10]))
    #plt.plot(centres*1e3, 1e9*measured_emittances[:, 20], 'r:')
    #plt.plot(centres*1e3, 1e9*corrected_emittance[:, 20], 'r', label='{:d} holes'.format(num_holess[20]))
    plt.plot(centres*1e3, 1e9*measured_emittances[:, -6], 'c:')
    plt.plot(centres*1e3, 1e9*corrected_emittance[:, -6], 'c', label='{:d}␣holes'.format(num_holess[-6]))
    plt.axhline(expected_e*1e9, ls='--', color='r')
    plt.xlabel('Centre␣(mm)')
    plt.ylabel('Emittance␣(nm␣rad)')
    plt.legend(loc='upper␣left')

    plt.figure('Normalised␣Off␣Centre␣Pepperpot')
    plt.title('Normalised␣Off␣Centre␣Pepperpot')
    cs = linspace(0, 2, 100)
    n_corrs = zeros((cs.size, num_holess.size))
    for i, c in enumerate(cs):
        for j, n in enumerate(num_holess):
            en = (n*pepperpot_pitch/2)/rms_beam_size_sample

            x_lo = (c - en)
            x_hi = (c + en)

            n_corrs[i, j] = normal_prop(x_lo, x_hi)

    corrected_emittance = corrections * measured_emittances
    plt.plot(centres/rms_beam_size_sample, measured_emittances[:, 0]/expected_e, 'b:')
    plt.plot(cs, n_corrs[:, 0], 'b', label='{:d}␣holes'.format(num_holess[0]))
    plt.plot(centres/rms_beam_size_sample, measured_emittances[:, 10]/expected_e, 'g:')
    plt.plot(cs, n_corrs[:, 10], 'g', label='{:d}␣holes'.format(num_holess[10]))
    #plt.plot(centres/rms_beam_size_sample, measured_emittances[:, 20]/expected_e, 'r:')
    #plt.plot(cs, n_corrs[:, 20], 'r', label='{:d} holes'.format(num_holess[20]))
    plt.plot(centres/rms_beam_size_sample, measured_emittances[:, -6]/expected_e, 'c:')
    plt.plot(cs, n_corrs[:, -6], 'c', label='{:d}␣holes'.format(num_holess[-6]))

    plt.axhline(1, ls='--', color='r')
    plt.xlabel('Centre␣(mm)')
    plt.ylabel('Emittance␣(nm␣rad)')
    plt.legend(loc='upper␣left')
elif True:
    # Plotting for single num_holes, many centres.
    correction_f = lambda centre, magic_number: measured_emittances[:, 0]/normal_prop( \
            (centre - num_holess[0]*pepperpot_pitch/2)*magic_number / rms_beam_size_sample, \
            (centre + num_holess[0]*pepperpot_pitch/2)*magic_number / rms_beam_size_sample)

    guess = 3/4

    fit = fitCurve(centres, zeros(centres.size)+expected_e, correction_f, guess)

    print(fit)
    fitted_magic = 0.75

    corrections = zeros(measured_emittances.shape)
    for i, c in enumerate(centres):
        for j, n in enumerate(num_holess):
            x_lo = (c - n*pepperpot_pitch/2)*(3/4) / rms_beam_size_sample
            x_hi = (c + n*pepperpot_pitch/2)*(3/4) / rms_beam_size_sample

            corrections[i, j] = 1/normal_prop(x_lo, x_hi)

    print(1e9*measured_emittances[:, 0]*corrections[:, 0])
    print(1e9*correction_f(centres, fitted_magic))

    plt.figure()
```

```
        plt.plot(centres*1e3, 1e9*measured_emittances[:, 0], ':', label='Raw')
        plt.plot(centres*1e3, 1e9*measured_emittances[:, 0]*corrections[:, 0], label='Corrected')
        plt.plot(centres*1e3, 1e9*correction_f(centres, fitted_magic), 'x', label='Fitted')
        plt.axhline(expected_e*1e9, ls='--', color='r')
        plt.legend(loc='upper left')
        plt.xlabel('Pepperpot Centre Offset (mm)')
        plt.ylabel('Emittance (nm rad)')

# Looking at coverage at a focus.
if False:
    filename = 'coverage focus.h5'

    if False:
        wavelengths = array([475e-9])#linspace(465e-9, 487e-9, 100)

        excess_energys = excess_energy_from_wavelength(wavelengths, field_ionisation=False)

        expected_emittances = expected_emittance(excess_energys, beam_size_rms)

        # Pepperpot
        number_of_holess = arange(2, 15, 1)
        centres = arange(0, 3, 0.5)*beam_size_rms
        number_of_holes = 10
        pepperpot_pitch = 400e-6
        aperture_size = 50e-6

        bins = arange(-2.5e-3, 2.5e-3, 5e-6)

        N = 1000000
        propagation_distance = 0.02
        expected_e = expected_emittances[0]

        base_bunch = Bunch(n=N, energy=beam_energy, rms_width=beam_size_rms, \
                           normalised_rms_emittance=expected_e, mass=m_e)

        # For each +'ve excess energy simulate a bunch.
        measured_emittances = zeros((centres.size, number_of_holess.size))
        measured_emittances_beam_size_corrected = zeros((centres.size, number_of_holess.size))
        for j, centre in enumerate(centres):
            for i, number_of_holes in enumerate(number_of_holess):
                if isnan(expected_e):
                    measured_emittances[i] = float('nan')
                    measured_emittances_beam_size_corrected[i] = float('nan')
                    continue

                print()
                print('\tEmittance: {:.2f}nm rad'.format(1e9*expected_e))
                print('\tCentre: {:.2f}um'.format(1e6*centre))
                print('\tNumber of holes:', number_of_holes)

                #pepperpot = pepperpotMask(pitch=pepperpot_pitch, number_holes=number_of_holes, pinhole_diameter=aperture_size)

                pepperpotFs = pepperpotMaskFs(pitch=pepperpot_pitch, number_holes=number_of_holes, pinhole_diameter=aperture_size, location

                bunch = base_bunch.copy()

                print('\tBunch RMS Size: {:.2f}um'.format(bunch.getWidth()*1e6))

                bunch = maskBunch2(bunch, pepperpotFs)

                bunch.propagate(propagation_distance)

                hist, bin_edges = histogram(bunch.getXs(), bins=bins)
                m_per_pixel = bin_edges[1]-bin_edges[0]
                hist = array(hist, dtype='f')

                peaks = find_peaks(bin_edges[:-1], hist, min_dist=60, diag=False)


                plt.figure()
                # Don't correct for beam size.
                emittance, rms_size, total_count = emittance_from_line(hist, peaks, sum_peaks=True, diag=True,
                                m_per_pixel=m_per_pixel, number_holes=len(peaks),
```

```
                        pitch=pepperpot_pitch, propagation_distance=propagation_distance,
                        adjust_for_size=False, adjust_for_aperture_size=True, refine_parameters=True)
            emittance *= bunch.getBeta()

            measured_emittances[j, i] = emittance

            # Correct for beam size.
            emittance, rms_size, total_count = emittance_from_line(hist, peaks, sum_peaks=True, diag=False,
                        m_per_pixel=m_per_pixel, number_holes=len(peaks),
                        pitch=pepperpot_pitch, propagation_distance=propagation_distance,
                        adjust_for_size=True, adjust_for_aperture_size=True, refine_parameters=True)
            emittance *= bunch.getBeta()

            measured_emittances_beam_size_corrected[j, i] = emittance

            print('\tMeasured Emittance: {:.2f}nm rad'.format(emittance*1e9))

            if False:
                plt.figure('hist')
                plt.plot(bin_edges[:-1], hist)

                plt.show()
                exit()


    # Save data
    if True:
        with File(filename) as hdf:
            number_of_holes_key = 'number of holes'
            if number_of_holes_key in hdf:
                del hdf[number_of_holes_key]

            hdf.create_dataset(number_of_holes_key, data=number_of_holess)

            centre_key = 'centres'
            if centre_key in hdf:
                del hdf[centre_key]

            hdf.create_dataset(centre_key, data=centres)


            measured_emittance_key = 'measured emittance'
            if measured_emittance_key in hdf:
                del hdf[measured_emittance_key]

            hdf.create_dataset(measured_emittance_key, data=measured_emittances)

            measured_emittance_key = 'measured emittance beam size correction'
            if measured_emittance_key in hdf:
                del hdf[measured_emittance_key]

            hdf.create_dataset(measured_emittance_key, data=measured_emittances_beam_size_corrected)

            hdf.attrs['wavelength'] = wavelengths[0]
            hdf.attrs['excess energy'] = excess_energys[0]
            hdf.attrs['expected emittance'] = expected_e
            hdf.attrs['aperture size'] = aperture_size
            hdf.attrs['pitch'] = pepperpot_pitch
else:
    with File(filename) as hdf:
        number_of_holess = array(hdf['number of holes'])
        centres = array(hdf['centres'])
        measured_emittances = array(hdf['measured emittance'])
        measured_emittances_beam_size_corrected = array(hdf['measured emittance beam size correction'])

        expected_e = hdf.attrs['expected emittance']
        aperture_size = hdf.attrs['aperture size']
        pepperpot_pitch = hdf.attrs['pitch']


# Plotting
for i, c in enumerate(centres):
    if False:
```

```
            plt.figure('Number of Holes vs. Emittance ' + str(c))
            plt.title('Number of Holes vs. Emittance ' + str(c))
            plt.plot(number_of_holess, measured_emittances[i]*1e9)
            #plt.plot(number_of_holess, measured_emittances_beam_size_corrected[i]*1e9)
            plt.axhline(expected_e*1e9, color='r', ls=':')
            plt.xlabel('Number of Holes')
            plt.ylabel('Emittance (nm rad)')

        plt.figure('Normalised ' + str(c))
        plt.title('Normalised ' + str(c))
        norm_xs = number_of_holess*pepperpot_pitch/beam_size_rms
        norm_ys = measured_emittances[i]/expected_e
        plt.plot(norm_xs, norm_ys)

        xs = arange(0.1, 15, 0.1)*pepperpot_pitch / beam_size_rms
        normalised_centre = c / beam_size_rms
        fit_f = lambda x, magic: normal_prop(-magic*(x-normalised_centre)/2, \
                                                magic*(x-normalised_centre)/2)
        guess = [0.75]
        fit = fitCurve(norm_xs, norm_ys, fit_f, guess)


        MAGIC = fit[0]
        ys = array([normal_prop(-MAGIC*(x-normalised_centre)/2, \
                                MAGIC*(x-normalised_centre)/2) for x in xs])
        plt.plot(xs, ys)

    if False:
        # Plot for thesis.
        rcParams.update({'font.size': 10})
        rcParams.update({'pgf.rcfonts': False})
        rcParams.update({'pgf.texsystem': 'pdflatex'})

        # Font should match document
        rcParams['font.family'] = 'serif'

        rcParams['axes.unicode_minus'] = False
            # Minus sign from matplot lib is too long for my taste.


        linewidth = 5.71 # inches

        figwidth = 0.95*linewidth
        figheight = figwidth/3*1.44
        figsize = (figwidth, figheight)

        plt.figure(figsize=figsize)
        plt.plot(excess_energys*1e3, measured_emittances*1e9, '-', markersize=5)
        plt.plot(excess_energys*1e3, measured_emittances_aperture_size_corrected*1e9, '-', markersize=5)
        plt.plot(excess_energys*1e3, expected_emittances*1e9, ':r')
        plt.xlabel('Excess Energy (meV)')
        plt.ylabel('Emittance (nm rad)')

        plt.tight_layout()

        plt.savefig('coverage at a focus.pgf')


# Experimenting with aperture size.
if True:
    file_name = 'pepperpot_aperture_size.h5'

    n = 10000000
    d_source_to_lens = .25+.450
    d_lens_to_sample = .100
    d_sample_to_detector = .45+.430

    lens_strength = -1.0e9

    pepperpot_pitch = 250e-6
    number_of_holes = 21

    if False:
```

```
# Simulate and Save.
wavelengths = array([475e-9])

excess_energys = excess_energy_from_wavelength(wavelengths, field_ionisation=False)

expected_emittances = expected_emittance(excess_energys, beam_size_rms)

expected_e = expected_emittances[0]

# To save time, propagate a bunch to the pepperpots and reuse.
save_time = True
if save_time:
    # Big bunch
    N = n*10
    print('Emittance: {:.2f}nm rad'.format(1e9*expected_e))
    big_base_bunch = Bunch(n=N, energy=beam_energy, rms_width=beam_size_rms, \
                  normalised_rms_emittance=expected_e, mass=m_e)
    print('Initial bunch emittance: {:.2f}nm rad'.format(big_base_bunch.getNormalisedRMSEmittance()*1e9))

    big_base_bunch.propagate(d_source_to_lens)
    simple_lens(big_base_bunch, strength=lens_strength)
    print('After lens bunch emittance: {:.2f}nm rad'.format(big_base_bunch.getNormalisedRMSEmittance()*1e9))
    big_base_bunch.propagate(d_lens_to_sample)
    print('Beam size at sample: {:.2f}mm'.format(big_base_bunch.getWidth()*1e3))
    rms_beam_size_sample = big_base_bunch.getWidth()

    # 'Small' bunch
    N = n
    print('Emittance: {:.2f}nm rad'.format(1e9*expected_e))
    small_base_bunch = Bunch(n=N, energy=beam_energy, rms_width=beam_size_rms, \
                  normalised_rms_emittance=expected_e, mass=m_e)
    print('Initial bunch emittance: {:.2f}nm rad'.format(small_base_bunch.getNormalisedRMSEmittance()*1e9))

    small_base_bunch.propagate(d_source_to_lens)
    simple_lens(small_base_bunch, strength=lens_strength)
    print('After lens bunch emittance: {:.2f}nm rad'.format(small_base_bunch.getNormalisedRMSEmittance()*1e9))
    small_base_bunch.propagate(d_lens_to_sample)
    print('Beam size at sample: {:.2f}mm'.format(small_base_bunch.getWidth()*1e3))
    rms_beam_size_sample = small_base_bunch.getWidth()

mn = -0.025
mx = 0.025
rn = mx - mn
bins = linspace(mn - rn, mx + rn, 1000)


aperture_sizes = linspace(1e-6, 100e-6, 10)
aperture_sizes = array([1e-6, 2e-6, 3e-6, 4e-6, 5e-6, 6e-6, 7e-6, 8e-6, 9e-6, 10e-6, \
                        20e-6, 30e-6, 40e-6, 50e-6, 60e-6, 70e-6, 80e-6, 90e-6, 100e-6])
measured_emittances = zeros(aperture_sizes.size)
measured_corrected_emittances = zeros(aperture_sizes.size)
measured_beam_size = zeros(aperture_sizes.size)
for k, aperture_size in enumerate(aperture_sizes):
    if not save_time:
        if aperture_size<10e-6:
            N = n*10
        else:
            N = n
        print('Emittance: {:.2f}nm rad'.format(1e9*expected_e))
        base_bunch = Bunch(n=N, energy=beam_energy, rms_width=beam_size_rms, \
                      normalised_rms_emittance=expected_e, mass=m_e)
        print('Initial bunch emittance: {:.2f}nm rad'.format(base_bunch.getNormalisedRMSEmittance()*1e9))

        base_bunch.propagate(d_source_to_lens)
        simple_lens(base_bunch, strength=lens_strength)
        print('After lens bunch emittance: {:.2f}nm rad'.format(base_bunch.getNormalisedRMSEmittance()*1e9))
        base_bunch.propagate(d_lens_to_sample)
        print('Beam size at sample: {:.2f}mm'.format(base_bunch.getWidth()*1e3))
        rms_beam_size_sample = base_bunch.getWidth()
    else:
        if aperture_size<10e-6:
            base_bunch = big_base_bunch
        else:
```

```
            base_bunch = small_base_bunch

        print()
        print('Pepperpot aperture size: {:.2f}um'.format(aperture_size*1e6))
        # Pepperpot
        pepperpot = pepperpotMask(pitch=pepperpot_pitch, number_holes=number_of_holes, \
                                  location=0, pinhole_diameter=aperture_size)

        bunch = base_bunch.copy()

        bunch = maskBunch(bunch, pepperpot)
        print('\tAfter sample bunch emittance: {:.2f}nm rad'.format(bunch.getNormalisedRMSEmittance()*1e9))
        print('\tAfter sample bunch count: {:d}'.format(bunch.getSize()))

        bunch.propagate(d_sample_to_detector)

        hist, bin_edges = histogram(bunch.getXs(), bins=bins)
        m_per_pixel = bin_edges[1]-bin_edges[0]

        hist = array(hist, dtype='f')

        plt.figure()
        peaks = find_peaks(bin_edges[:-1], hist, min_dist=25, diag=True, smooth=True)


        plt.figure()
        emittance, rms_size, total_count = emittance_from_line(hist, peaks, sum_peaks=True, diag=True,
                        m_per_pixel=m_per_pixel, number_holes=len(peaks),
                        pitch=pepperpot_pitch, propagation_distance=d_sample_to_detector,
                        adjust_for_size=True, adjust_for_aperture_size=False, refine_parameters=True)
        emittance *= bunch.getBeta()

        corrected_emittance, rms_size, total_count = emittance_from_line(hist, peaks, sum_peaks=True, \
                        diag=True, m_per_pixel=m_per_pixel, number_holes=len(peaks),
                        pitch=pepperpot_pitch, propagation_distance=d_sample_to_detector,
                        aperture_size=aperture_size, adjust_for_size=True, \
                        adjust_for_aperture_size=True, refine_parameters=True)
        corrected_emittance *= bunch.getBeta()

        measured_emittances[k] = emittance
        measured_corrected_emittances[k] = corrected_emittance
        measured_beam_size[k] = rms_size

        print('\tMeasured Emittance: {:.2f}nm rad'.format(emittance*1e9))


    # Save data
    with File(file_name) as hdf:
        aperture_size_key = 'aperture size'
        if aperture_size_key in hdf:
            del hdf[aperture_size_key]

        hdf.create_dataset(aperture_size_key, data=aperture_sizes)

        measured_emittance_key = 'measured emittance'
        if measured_emittance_key in hdf:
            del hdf[measured_emittance_key]

        hdf.create_dataset(measured_emittance_key, data=measured_emittances)


        measured_corrected_emittance_key = 'measured corrected emittance'
        if measured_corrected_emittance_key in hdf:
            del hdf[measured_corrected_emittance_key]

        hdf.create_dataset(measured_corrected_emittance_key, data=measured_corrected_emittances)

        measured_beam_size_key = 'measured beam size'
        if measured_beam_size_key in hdf:
            del hdf[measured_beam_size_key]

        hdf.create_dataset(measured_beam_size_key, data=measured_beam_size)
```

```
            hdf.attrs['expected emittance'] = expected_e
            hdf.attrs['rms beam width at pepperpot'] = rms_beam_size_sample
    else:
        # Load data
        with File(file_name) as hdf:
            expected_e = hdf.attrs['expected emittance']
            rms_beam_size_sample = hdf.attrs['rms beam width at pepperpot']
            aperture_sizes = array(hdf['aperture size'])
            measured_emittances = array(hdf['measured emittance'])
            measured_corrected_emittances = array(hdf['measured corrected emittance'])

    # Plot Stuff
    plt.figure('Aperture Sizes vs Measured Emittance')

    plt.subplot(2, 1, 1)
    plt.title('Raw')
    plt.plot(aperture_sizes*1e6, measured_emittances*1e9)
    plt.plot(aperture_sizes*1e6, measured_corrected_emittances*1e9)
    plt.xlabel('Aperture Size (um)')
    plt.ylabel('Measured Emittance (nm rad)')
    plt.axhline(expected_e*1e9, ls=':', color='k')

    plt.subplot(2, 1, 2)
    plt.title('Normalised')
    plt.plot(aperture_sizes*1e6, measured_emittances/expected_e)
    plt.plot(aperture_sizes*1e6, measured_corrected_emittances/expected_e)
    plt.xlabel('Aperture Size (um)')
    plt.ylabel('Measured Emittance/Expected Emittance')
    plt.axhline(1, ls=':', color='k')

    if True:
        # Plot for thesis.
        rcParams.update({'font.size': 10})
        rcParams.update({'pgf.rcfonts': False})
        rcParams.update({'pgf.texsystem': 'pdflatex'})

        # Font should match document
        rcParams['font.family'] = 'serif'

        rcParams['axes.unicode_minus'] = False
            # Minus sign from matplot lib is too long for my taste.


        colours = [(79/255,122/255,174/255),(255/255,102/255,51/255),(245/255,174/255,32/255),(77/255,155/255,77/255),(102/255,102/255,102/255)


        linewidth = 5.71 # inches

        figwidth = linewidth
        figheight = figwidth/3
        figwidth *= 0.75
        figsize = (figwidth, figheight)

        plt.figure(figsize=figsize)
        plt.plot(aperture_sizes*1e6, measured_emittances*1e9, color=colours[0])
        plt.plot(aperture_sizes*1e6, measured_corrected_emittances*1e9, color=colours[3])
        plt.xlabel('Aperture Size ($\mu$m)')
        plt.ylabel('Emittance (nm rad)')
        plt.axhline(expected_e*1e9, ls=':', color='k')

        plt.tight_layout()

        plt.savefig('aperture_size_sim.pgf')


# Looking at overlapping beamlets.
if False:
    file_name = 'pepperpot_overlap.h5'

    N = 10000000
    d_source_to_lens = .25 + .450
    d_lens_to_sample = .100
    d_sample_to_detector = .45 + .430
```

```
lens_strength = -1.0e9

pepperpot_pitch = 200e-6
number_of_holes = 30
aperture_size = 50e-6

if True:
    # Simulate and Save.
    excess_energys = linspace(-5e-3, 100e-3, 20)

    excess_energys = excess_energys[-3:-1]

    expected_emittances = expected_emittance(excess_energys, beam_size_rms)

    mn = -0.035
    mx = 0.035
    rn = mx - mn
    bins = linspace(mn - rn, mx + rn, 2000)

    measured_emittances = zeros(expected_emittances.size)
    measured_emittances_not_corrected = zeros(expected_emittances.size)
    measured_beam_size = zeros(expected_emittances.size)
    for k, expected_e in enumerate(expected_emittances):
        if isnan(expected_e):
            expected_e = 0

        print('Emittance: {:.2f}nm rad'.format(1e9*expected_e))
        base_bunch = Bunch(n=N, energy=beam_energy, rms_width=beam_size_rms_cherry, \
                    normalised_rms_emittance=expected_e, mass=m_e)
        print('Initial bunch emittance: {:.2f}nm rad'.format(base_bunch.getNormalisedRMSEmittance()*1e9))

        base_bunch.propagate(d_source_to_lens)
        simple_lens(base_bunch, strength=lens_strength)
        print('After lens bunch emittance: {:.2f}nm rad'.format(base_bunch.getNormalisedRMSEmittance()*1e9))
        base_bunch.propagate(d_lens_to_sample)
        rms_beam_size_sample = base_bunch.getWidth()
        print('Beam size at sample: {:.2f}um'.format(rms_beam_size_sample*1e6))

        print()

        # Pepperpot
        pepperpot = pepperpotMask(pitch=pepperpot_pitch, number_holes=number_of_holes, \
                                location=0, pinhole_diameter=aperture_size)

        bunch = base_bunch.copy()

        bunch = maskBunch(bunch, pepperpot)
        print('\tAfter sample bunch emittance: {:.2f}nm rad'.format(bunch.getNormalisedRMSEmittance()*1e9))
        print('\tAfter sample bunch count: {:d}'.format(bunch.getSize()))

        bunch.propagate(d_sample_to_detector)

        hist, bin_edges = histogram(bunch.getXs(), bins=bins)
        m_per_pixel = bin_edges[1]-bin_edges[0]

        peaks = find_peaks(bin_edges[:-1], hist, min_dist=25, diag=True, smooth=True)

        hist = array(hist, dtype='f')

        plt.figure()
        emittance, rms_size, _total_count = emittance_from_line(hist, peaks, sum_peaks=True, diag=True,
                        m_per_pixel=m_per_pixel, number_holes=len(peaks),
                        pitch=pepperpot_pitch, propagation_distance=d_sample_to_detector,
                        adjust_for_size=True, refine_parameters=True, adjust_for_aperture_size=True)
        emittance *= bunch.getBeta()

        plt.figure()
        emittance2, _rms_size, _total_count = emittance_from_line(hist, peaks, sum_peaks=True, diag=True,
                        m_per_pixel=m_per_pixel, number_holes=len(peaks),
                        pitch=pepperpot_pitch, propagation_distance=d_sample_to_detector,
                        adjust_for_size=True, refine_parameters=False, adjust_for_aperture_size=True)
        emittance2 *= bunch.getBeta()
```

```
            measured_emittances[k] = emittance
            measured_emittances_not_corrected[k] = emittance2
            measured_beam_size[k] = rms_size

            print('\tMeasured Emittance: {:.2f}nm rad'.format(emittance*1e9))
            print('\tMeasured Emittance (not corrected): {:.2f}nm rad'.format(emittance2*1e9))
            print()
            print()

            if False:
                plt.show()
                exit()

        # Save data
        if True:
            with File(file_name) as hdf:
                measured_emittance_key = 'measured emittance'
                if measured_emittance_key in hdf:
                    del hdf[measured_emittance_key]

                hdf.create_dataset(measured_emittance_key, data=measured_emittances)

                measured_emittance_nt_corrected_key = 'measured emittance nor corrected'
                if measured_emittance_nt_corrected_key in hdf:
                    del hdf[measured_emittance_nt_corrected_key]

                hdf.create_dataset(measured_emittance_nt_corrected_key, data=measured_emittances_not_corrected)

                measured_beam_size_key = 'measured beam size'
                if measured_beam_size_key in hdf:
                    del hdf[measured_beam_size_key]

                hdf.create_dataset(measured_beam_size_key, data=measured_beam_size)

                expected_emittance_key = 'expected emittance'
                if expected_emittance_key in hdf:
                    del hdf[expected_emittance_key]

                hdf.create_dataset(expected_emittance_key, data=expected_emittances)

                excess_energy_key = 'excess energy'
                if excess_energy_key in hdf:
                    del hdf[excess_energy_key]

                hdf.create_dataset(excess_energy_key, data=excess_energys)

    else:
        # Load data
        with File(file_name) as hdf:
            excess_energys = array(hdf['excess energy'])
            expected_emittances = array(hdf['expected emittance'])

            measured_emittances = array(hdf['measured emittance'])
            measured_emittances_not_corrected = array(hdf['measured emittance nor corrected'])


    if True:
        # Plot
        plt.figure('Beamlet overlap')

        plt.plot(excess_energys*1e3, measured_emittances*1e9)
        plt.plot(excess_energys*1e3, measured_emittances_not_corrected*1e9)
        plt.plot(excess_energys*1e3, expected_emittances*1e9)

# Looking at overlapping beamlets. Take 2.
if False:
    beam_size = beam_size_rms_cherry
    filename = 'overlap.h5'

    if True:
        # Simulate and save
        wavelengths = linspace(445e-9, 487e-9, 100)[11:12]
```

```
excess_energys = excess_energy_from_wavelength(wavelengths, field_ionisation=False)

expected_emittances = expected_emittance(excess_energys, beam_size)

# Pepperpot
number_of_holes = 9
pepperpot_pitch = 150e-6
aperture_size = 50e-6
pepperpot = pepperpotMask(pitch=pepperpot_pitch, number_holes=number_of_holes, pinhole_diameter=aperture_size)

# Bins for histogram 'detector'
if False:
    bins = 1000
else:
    bins = linspace(-.001, 0.001, 2000)

N = 10000000
propagation_distance = 0.015
# For each +'ve excess energy simulate a bunch.
measured_emittances = zeros(wavelengths.size)
measured_emittances_beam_size_corrected = zeros(wavelengths.size)
measured_emittances_aperture_size_corrected = zeros(wavelengths.size)
for i, expected_e in enumerate(expected_emittances):
    if isnan(expected_e):
        if False:
            # Skip
            measured_emittances[i] = float('nan')
            measured_emittances_aperture_size_corrected[i] = float('nan')
            continue
        else:
            # Use zero emittance beam.
            expected_e = 0
            expected_emittances[i] = expected_e

    print(i)
    print('Emittance:␣{:.2f}nm␣rad'.format(1e9*expected_e))

    bunch = Bunch(n=N, energy=beam_energy, rms_width=beam_size, \
                  normalised_rms_emittance=expected_e, mass=m_e)

    print('\tBunch␣RMS␣Size:␣{:.2f}um'.format(bunch.getWidth()*1e6))

    bunch = maskBunch(bunch, pepperpot)

    bunch.propagate(propagation_distance)

    hist, bin_edges = histogram(bunch.getXs(), bins=bins)
    m_per_pixel = bin_edges[1]-bin_edges[0]
    hist = array(hist, dtype='f')

    peaks = find_peaks(bin_edges[:-1], hist, thres=0.005, min_dist=100, diag=True, smooth=True, smooth_size=10)


    if True:
        # Try smoothing the histogram.
        hist = gaussian_filter(hist, 5)

    # Don't correct for anything.
    emittance, rms_size, total_count = emittance_from_line(hist, peaks, sum_peaks=True, diag=False,
                    m_per_pixel=m_per_pixel, number_holes=len(peaks),
                    pitch=pepperpot_pitch, propagation_distance=propagation_distance,
                    adjust_for_size=False, adjust_for_aperture_size=False, refine_parameters=False)
    emittance *= bunch.getBeta()

    measured_emittances[i] = emittance

    print('\tMeasured␣Emittance␣(no␣corrections):␣␣␣␣␣␣␣␣{:.2f}nm␣rad'.format(emittance*1e9))

    # Don't correct for overlap.
    emittance, rms_size, total_count = emittance_from_line(hist, peaks, sum_peaks=True, diag=False,
                    m_per_pixel=m_per_pixel, number_holes=len(peaks),
                    pitch=pepperpot_pitch, propagation_distance=propagation_distance,
```

```
                adjust_for_size=True, adjust_for_aperture_size=True, refine_parameters=False)
    emittance *= bunch.getBeta()

    measured_emittances_beam_size_corrected[i] = emittance

    print('\tMeasured Emittance (no overlap correction): {:.2f}nm rad'.format(emittance*1e9))

    # Correct for everything.
    plt.figure()
    emittance, _rms_size, _total_count = emittance_from_line(hist, peaks, sum_peaks=True, diag=True,
                m_per_pixel=m_per_pixel, number_holes=len(peaks),
                pitch=pepperpot_pitch, propagation_distance=propagation_distance,
                adjust_for_size=True, adjust_for_aperture_size=True, refine_parameters=True)
    emittance *= bunch.getBeta()
    measured_emittances_aperture_size_corrected[i] = emittance

    print('\tMeasured Emittance (all corrections):       {:.2f}nm rad'.format(emittance*1e9))

    with File('overlap_example.h5') as hdf:
        histogram_key = 'histogram'
        hdf.create_dataset(histogram_key, data=hist)

        hdf.attrs['expected emittance'] = expected_e
        hdf.attrs['excess energy'] = excess_energys[0]
        hdf.attrs['wavelength'] = wavelengths[0]
        hdf.attrs['source size'] = beam_size
        hdf.attrs['no correction emittance'] = measured_emittances[i]
        hdf.attrs['corrected unrefined emittance'] = measured_emittances_beam_size_corrected[i]
        hdf.attrs['corrected refined emittance'] = measured_emittances_aperture_size_corrected[i]
        hdf.attrs['m_per_pixel'] = m_per_pixel

    if True:
        plt.figure('hist')
        plt.plot(bin_edges[:-1], hist)

        plt.show()
        exit()


# Save data
if True:
    with File(filename) as hdf:
        wavelengths_key = 'wavelengths'
        if wavelengths_key in hdf:
            del hdf[wavelengths_key]

        hdf.create_dataset(wavelengths_key, data=wavelengths)

        excess_energy_key = 'excess energy'
        if excess_energy_key in hdf:
            del hdf[excess_energy_key]

        hdf.create_dataset(excess_energy_key, data=excess_energys)

        expected_emittance_key = 'expected emittance'
        if expected_emittance_key in hdf:
            del hdf[expected_emittance_key]

        hdf.create_dataset(expected_emittance_key, data=expected_emittances)

        measured_emittance_key = 'measured emittance'
        if measured_emittance_key in hdf:
            del hdf[measured_emittance_key]

        hdf.create_dataset(measured_emittance_key, data=measured_emittances)

        measured_emittance_key = 'measured emittance beam size correction'
        if measured_emittance_key in hdf:
            del hdf[measured_emittance_key]

        hdf.create_dataset(measured_emittance_key, data=measured_emittances_beam_size_corrected)

        measured_emittance_key = 'measured emittance aperture size correction'
```

```
            if measured_emittance_key in hdf:
                del hdf[measured_emittance_key]

            hdf.create_dataset(measured_emittance_key, data=measured_emittances_aperture_size_corrected)

            hdf.attrs['number of holes'] = number_of_holes
            hdf.attrs['aperture size'] = aperture_size
            hdf.attrs['pitch'] = pepperpot_pitch
else:
    with File(filename) as hdf:
        wavelengths = array(hdf['wavelengths'])
        excess_energys = array(hdf['excess energy'])
        expected_emittances = array(hdf['expected emittance'])
        measured_emittances = array(hdf['measured emittance'])
        measured_emittances_beam_size_corrected = array(hdf['measured emittance beam size correction'])
        measured_emittances_aperture_size_corrected = array(hdf['measured emittance aperture size correction'])

        number_of_holes = hdf.attrs['number of holes']
        aperture_size = hdf.attrs['aperture size']
        pepperpot_pitch = hdf.attrs['pitch']


# Plotting
plt.figure('Wavelength vs. Excess Energy')
plt.title('Wavelength vs. Excess Energy')
plt.plot(wavelengths*1e9, excess_energys*1e3)
plt.xlabel('Wavelength (nm)')
plt.ylabel('Excess Energy (meV)')
plt.axhline(0, color='k', ls=':')
plt.xlim((wavelengths.min()*1e9, wavelengths.max()*1e9))


plt.figure('Excess Emittance vs. Emittance')
plt.title('Excess Emittance vs. Emittance')
plt.plot(excess_energys*1e3, expected_emittances*1e9, 'r')
plt.plot(excess_energys*1e3, measured_emittances*1e9, 'bx')
plt.plot(excess_energys*1e3, measured_emittances_beam_size_corrected*1e9, 'gx')
plt.plot(excess_energys*1e3, measured_emittances_aperture_size_corrected*1e9, 'kx')
plt.xlabel('Excess Energy (meV)')
plt.ylabel('Emittance (nm rad)')
plt.xlim((0, excess_energys.max()*1e3))

if False:
    # Plot for thesis.
    rcParams.update({'font.size': 10})
    rcParams.update({'pgf.rcfonts': False})
    rcParams.update({'pgf.texsystem': 'pdflatex'})

    colours = [(79/255,122/255,174/255),(255/255,102/255,51/255),(245/255,174/255,32/255),(77/255,155/255,77/255),(102/255,102/255,102/

    # Font should match document
    rcParams['font.family'] = 'serif'

    rcParams['axes.unicode_minus'] = False
        # Minus sign from matplot lib is too long for my taste.


    linewidth = 5.71 # inches

    figwidth = linewidth
    figheight = figwidth/3
    figwidth *= 0.75
    figsize = (figwidth, figheight)

    # Higher res theory
    wavelengths_fine = linspace(465e-9, 487e-9, 10000)

    excess_energys_fine = excess_energy_from_wavelength(wavelengths_fine, field_ionisation=False)

    expected_emittances_fine = expected_emittance(excess_energys_fine, beam_size)
    for i in range(expected_emittances_fine.size):
        if isnan(expected_emittances_fine[i]):
            expected_emittances_fine[i] = 0
```

```
        # Plot
        plt.figure(figsize=figsize)
        plt.plot(excess_energys*1e3, measured_emittances*1e9, '-', markersize=5, color=colours[0])
        plt.plot(excess_energys*1e3, measured_emittances_aperture_size_corrected*1e9, '-', markersize=5, color=colours[3])
        plt.plot(excess_energys_fine*1e3, expected_emittances_fine*1e9, ':', color=colours[1])
        plt.xlim((-5, excess_energys.max()*1e3))
        plt.xlabel('Excess Energy (meV)')
        plt.ylabel('Emittance (nm rad)')

        plt.tight_layout()

        plt.savefig('wavelength_sweep_sim.pgf')


# Realistic Parameters - Extensive look at corrective factors.
if False:
    hdf_name = 'pepperpot_correction.h5'

    N = 10000000
    d_source_to_lens = .25+.450
    d_lens_to_sample = .100
    d_sample_to_detector = .45+.430

    lens_strength = -1.0e9



    if False:
        # Simulate and Save.
        wavelength = 475e-9
        excess_energy = excess_energy_from_wavelength(wavelength, field_ionisation=False)
        expected_emittance = expected_emittance(excess_energy, beam_size_rms)

        # To save time, propagate a bunch to the pepperpots and reuse.
        print('Emittance: {:.2f}nm rad'.format(1e9*expected_emittance))
        base_bunch = Bunch(n=N, energy=beam_energy, rms_width=beam_size_rms, \
                     normalised_rms_emittance=expected_emittance, mass=m_e)
        print('Initial bunch emittance: {:.2f}nm rad'.format(base_bunch.getNormalisedRMSEmittance()*1e9))

        base_bunch.propagate(d_source_to_lens)
        simple_lens(base_bunch, strength=lens_strength)
        print('After lens bunch emittance: {:.2f}nm rad'.format(base_bunch.getNormalisedRMSEmittance()*1e9))
        base_bunch.propagate(d_lens_to_sample)
        print('Beam size at sample: {:.2f}mm'.format(base_bunch.getWidth()*1e3))
        rms_beam_size_sample = base_bunch.getWidth()

        mn = -0.025
        mx = 0.025
        rn = mx - mn
        bins = linspace(mn - rn, mx + rn, 1000)

        # Pepperpot parameters
        pepperpot_pitchs = linspace(200e-6, 500e-6, 10)
        centres = linspace(0, 2*rms_beam_size_sample, 10)
        num_holess = array(flipud(arange(5, 20, 3)))


        measured_emittances = zeros((pepperpot_pitchs.size, centres.size, num_holess.size))
        measured_beam_size = zeros((pepperpot_pitchs.size, centres.size, num_holess.size))
        for i, pepperpot_pitch in enumerate(pepperpot_pitchs):
            for k, pepperpot_center in enumerate(centres):
                for j, num_holes in enumerate(num_holess):
                    print('{:d} of {:d}, {:d} of {:d}, {:d} of {:d}'.format(i, pepperpot_pitchs.size, \
                                                                            k, centres.size, \
                                                                            j, num_holess.size))
                    # Pepperpot
                    number_of_holes = num_holes
                    pepperpot = pepperpotMask(pitch=pepperpot_pitch, number_holes=number_of_holes, \
                                              location=pepperpot_center)

                    print('\tPepperpot Pitch: {:.2f}um'.format(pepperpot_pitch*1e6))
                    print('\tPepperpot Centre: {:.2f}um'.format(pepperpot_center*1e6))
```

```
                print('\tNumber of Holes:', num_holes)

                bunch = base_bunch.copy()

                bunch = maskBunch(bunch, pepperpot)

                bunch.propagate(d_sample_to_detector)

                hist, bin_edges = histogram(bunch.getXs(), bins=bins)
                m_per_pixel = bin_edges[1]-bin_edges[0]

                peaks = find_peaks(bin_edges[:-1], hist, min_dist=25, diag=False, smooth=True)

                hist = array(hist, dtype='f')

                if False:
                    plt.figure()
                    plt.plot(hist)
                    plt.show()

                emittance, rms_size, total_count = emittance_from_line(hist, peaks, \
                               sum_peaks=True, diag=False,
                               m_per_pixel=m_per_pixel, number_holes=len(peaks),
                               pitch=pepperpot_pitch, propagation_distance=d_sample_to_detector, \
                               adjust_for_aperture_size=True, refine_parameters=True, \
                               adjust_for_size=False)
                emittance *= bunch.getBeta()

                measured_emittances[i, k, j] = emittance
                measured_beam_size[i, k, j] = rms_size

                print('\tMeasured Emittance: {:.2f}nm rad'.format(emittance*1e9))


    # Save data
    with File(hdf_name) as hdf:
        pepperpot_pitch_key = 'pepperpot pitch'
        if pepperpot_pitch_key in hdf:
            del hdf[pepperpot_pitch_key]

        hdf.create_dataset(pepperpot_pitch_key, data=pepperpot_pitchs)

        number_of_holes_key = 'number of holes'
        if number_of_holes_key in hdf:
            del hdf[number_of_holes_key]

        hdf.create_dataset(number_of_holes_key, data=num_holess)

        centres_key = 'pepperpot centre'
        if centres_key in hdf:
            del hdf[centres_key]

        hdf.create_dataset(centres_key, data=centres)

        measured_emittance_key = 'measured emittance'
        if measured_emittance_key in hdf:
            del hdf[measured_emittance_key]

        hdf.create_dataset(measured_emittance_key, data=measured_emittances)

        measured_beam_size_key = 'measured beam size'
        if measured_beam_size_key in hdf:
            del hdf[measured_beam_size_key]

        hdf.create_dataset(measured_beam_size_key, data=measured_beam_size)

        hdf.attrs['expected emittance'] = expected_emittance
        hdf.attrs['rms beam width at pepperpot'] = rms_beam_size_sample
        hdf.attrs['number of electrons'] = N
else:
    # Load data
    with File(hdf_name) as hdf:
        expected_emittance = hdf.attrs['expected emittance']
```

```
            rms_beam_size_sample = hdf.attrs['rms beam width at pepperpot']

            num_holess = array(hdf['number of holes'])
            centres = array(hdf['pepperpot centre'])
            pepperpot_pitchs = array(hdf['pepperpot pitch'])

            measured_emittances = array(hdf['measured emittance'])
            measured_beam_size = array(hdf['measured beam size'])

    if True:
        # Plot

        corrections = zeros(measured_emittances.shape)
        for i, pitch in enumerate(pepperpot_pitchs):
            for k, c in enumerate(centres):
                for j, n in enumerate(num_holess):
                    x_lo = (c - n*pitch/2)*1 / rms_beam_size_sample
                    x_hi = (c + n*pitch/2)*1 / rms_beam_size_sample

                    corrections[i, k, j] = 1/normal_prop(x_lo, x_hi)


        # Pepperpot centre vs. emittance
        plt.figure()

        plt.axhline(expected_emittance*1e9, ls=':', color='r')

        for i, pitch in enumerate(pepperpot_pitchs):
            plt.plot(centres*1e3, 1e9*measured_emittances[i, :, 0])
            #plt.plot(centres*1e3, 1e9*measured_emittances[i, :, 0]*corrections[i, :, 0], 'x')

            correction_f = lambda centre, magic_number: measured_emittances[i, :, 0]/normal_prop( \
                (centre - num_holess[0]*pitch/2)*magic_number / rms_beam_size_sample, \
                (centre + num_holess[0]*pitch/2)*magic_number / rms_beam_size_sample)

            guess = 3/4

            fit = fitCurve(centres, zeros(centres.size)+expected_emittance, correction_f, guess)

            print(fit)

            plt.plot(centres*1e3, 1e9*correction_f(centres, 0.75), 'x')

        plt.xlabel('Pepperpot Centre (mm)')
        plt.ylabel('Measured Emittance (nm rad)')


#  Looking at beam coverage
if True:
    file_name = 'pepperpot_coverage.h5'

    N = 10000000
    d_source_to_lens = .25 + .450
    d_lens_to_sample = .100
    d_sample_to_detector = .45 + .430

    lens_strength = -1.0e9

    centres = array([0])#, 1, 2]) # Real beam RMS widths
    pepperpot_pitchs = array([200e-6, 250e-6, 300e-6])
    number_of_holess = arange(3, 31, 1)
    aperture_size = 50e-6

    if False:
        # Simulate and Save.
        source_size = beam_size_rms

        excess_energys = array([30e-3])

        expected_emittances = expected_emittance(excess_energys, source_size)

        expected_e = expected_emittances[0]
```

```
mn = -0.025
mx = 0.025
rn = mx - mn
bins = linspace(mn - rn, mx + rn, 1000)


print('Emittance: {:.2f}nm rad'.format(1e9*expected_e))
base_bunch = Bunch(n=N, energy=beam_energy, rms_width=source_size, \
             normalised_rms_emittance=expected_e, mass=m_e)
print('Initial bunch emittance: {:.2f}nm rad'.format(base_bunch.getNormalisedRMSEmittance()*1e9))

base_bunch.propagate(d_source_to_lens)
simple_lens(base_bunch, strength=lens_strength)
print('After lens bunch emittance: {:.2f}nm rad'.format(base_bunch.getNormalisedRMSEmittance()*1e9))
base_bunch.propagate(d_lens_to_sample)
rms_beam_size_sample = base_bunch.getWidth()
print('Beam size at sample: {:.2f}um'.format(rms_beam_size_sample*1e6))

print()

measured_emittances = zeros((centres.size, pepperpot_pitchs.size, number_of_holess.size))
measured_emittances_corrected = zeros((centres.size, pepperpot_pitchs.size, number_of_holess.size))
measured_beam_size = zeros((centres.size, pepperpot_pitchs.size, number_of_holess.size))
for m, centre in enumerate(centres):
    for j, pitch in enumerate(pepperpot_pitchs):
        for k, number_of_holes in enumerate(number_of_holess):
            print('Centre:', centre)
            print('Pitch: {:.2f}um'.format(pitch*1e6))
            print('Number of holes:', number_of_holes)
            # Pepperpot
            pepperpot = pepperpotMask(pitch=pitch, number_holes=number_of_holes, \
                                      location=centre*rms_beam_size_sample, \
                                      pinhole_diameter=aperture_size)

            bunch = base_bunch.copy()

            bunch = maskBunch(bunch, pepperpot)
            print('\tAfter sample bunch emittance: {:.2f}nm rad'.format(bunch.getNormalisedRMSEmittance()*1e9))
            print('\tAfter sample bunch count: {:d}'.format(bunch.getSize()))

            bunch.propagate(d_sample_to_detector)

            hist, bin_edges = histogram(bunch.getXs(), bins=bins)
            m_per_pixel = bin_edges[1]-bin_edges[0]

            peaks = find_peaks(bin_edges[:-1], hist, min_dist=25, diag=True, smooth=True)

            hist = array(hist, dtype='f')

            plt.figure()
            emittance, rms_size, total_count = emittance_from_line(hist, peaks, sum_peaks=True, \
                           diag=True, m_per_pixel=m_per_pixel, number_holes=len(peaks),
                           pitch=pitch, propagation_distance=d_sample_to_detector,
                           adjust_for_size=False, refine_parameters=True, adjust_for_aperture_size=True)
            emittance *= bunch.getBeta()

            emittance2, rms_size, total_count = emittance_from_line(hist, peaks, sum_peaks=True, \
                           diag=True, m_per_pixel=m_per_pixel, number_holes=len(peaks),
                           pitch=pitch, propagation_distance=d_sample_to_detector,
                           adjust_for_size=True, refine_parameters=True, adjust_for_aperture_size=True)
            emittance2 *= bunch.getBeta()

            measured_emittances_corrected[m, j, k] = emittance2
            measured_emittances[m, j, k] = emittance
            measured_beam_size[m, j, k] = rms_size

            print('\tMeasured Beam RMS Size: {:.2f}um'.format(rms_size*1e6))
            print('\tMeasured Emittance: {:.2f}nm rad'.format(emittance*1e9))
            print('\tMeasured Emittance (corrected): {:.2f}nm rad'.format(emittance2*1e9))
            print()

# Save data
if True:
```

```
        with File(file_name) as hdf:
            number_of_holes_key = 'number of holes'
            if number_of_holes_key in hdf:
                del hdf[number_of_holes_key]

            hdf.create_dataset(number_of_holes_key, data=number_of_holess)

            measured_emittance_key = 'measured emittance'
            if measured_emittance_key in hdf:
                del hdf[measured_emittance_key]

            hdf.create_dataset(measured_emittance_key, data=measured_emittances)

            measured_emittance_corrected_key = 'measured emittance corrected'
            if measured_emittance_corrected_key in hdf:
                del hdf[measured_emittance_corrected_key]

            hdf.create_dataset(measured_emittance_corrected_key, data=measured_emittances_corrected)

            measured_beam_size_key = 'measured beam size'
            if measured_beam_size_key in hdf:
                del hdf[measured_beam_size_key]

            hdf.create_dataset(measured_beam_size_key, data=measured_beam_size)

            hdf.attrs['expected emittance'] = expected_e
            hdf.attrs['rms beam width at pepperpot'] = rms_beam_size_sample
else:
    # Load data
    with File(file_name) as hdf:
        number_of_holess = array(hdf['number of holes'])
        expected_e = hdf.attrs['expected emittance']
        rms_beam_size_sample = hdf.attrs['rms beam width at pepperpot']
        measured_emittances = array(hdf['measured emittance'])
        measured_emittances_corrected = array(hdf['measured emittance corrected'])

if True:
    # Plot
    for j in range(centres.size):
        for i in range(pepperpot_pitchs.size):
            plt.figure('Raw ' + str(j) + ' Centre: {:.2f}'.format(centres[j]))
            plt.plot(number_of_holess, measured_emittances[j, i]*1e9, label='{:.2f}um Pitch'.format(pepperpot_pitchs[i]*1e6))

            plt.figure('Normalised ' + str(j))
            norm_xs = number_of_holess*pepperpot_pitchs[i] / rms_beam_size_sample
            norm_ys = measured_emittances[j, i]/expected_e
            plt.plot(norm_xs, norm_ys)

            fit_f = lambda x, magic: normal_prop(-magic*(x-centres[j]*rms_beam_size_sample)/2, \
                                                 magic*(x-centres[j]*rms_beam_size_sample)/2)
            guess = [0.77]

            fit = fitCurve(norm_xs, norm_ys, fit_f, guess)
            print('Pitch: {:.2f}um, Magic Number: {:.2f}'.format(pepperpot_pitchs[i]*1e6, fit[0]))



    for j in range(centres.size):

        xs = arange(0.1, 30, 0.1)*pepperpot_pitchs[-1] / rms_beam_size_sample
        MAGIC = 0.75
        ys = array([normal_prop(-MAGIC*(x-centres[j]*rms_beam_size_sample-aperture_size/2)/2, \
                                MAGIC*(x-centres[j]*rms_beam_size_sample+aperture_size/2)/2) \
                                for x in xs])

        MAGIC = 1
        ys2 = array([normal_prop(-MAGIC*(x-centres[j]*rms_beam_size_sample-aperture_size/2)/2, \
                                MAGIC*(x-centres[j]*rms_beam_size_sample+aperture_size/2)/2) for x in xs])

        plt.figure('Normalised ' + str(j))
        plt.plot(xs, ys, '--')
        plt.plot(xs, ys2, ':')
```

```
    plt.xlabel('RMS␣Widths')
    plt.ylabel('Emittance␣/␣Expected␣Emittance')

    plt.figure('Raw␣' + str(j) + '␣Centre:␣{:.2f}'.format(centres[j]))
    plt.axhline(expected_e*1e9, color='r', ls=':')
    plt.xlabel('Number␣of␣Holes')
    plt.ylabel('Emittance␣(nm␣rad)')
    plt.legend(loc='lower␣right')

if True:
    # Thesis plot
    rcParams.update({'font.size': 10})
    rcParams.update({'pgf.rcfonts': False})
    rcParams.update({'pgf.texsystem': 'pdflatex'})

    colours = [(79/255,122/255,174/255),(255/255,102/255,51/255),(245/255,174/255,32/255),(77/255,155/255,77/255),(102/255,102/255

    # Font should match document
    rcParams['font.family'] = 'serif'

    rcParams['axes.unicode_minus'] = False
        # Minus sign from matplot lib is too long for my taste.


    linewidth = 5.71 # inches

    figwidth = linewidth
    figheight = figwidth/3
    figwidth *= 0.75
    figsize = (figwidth, figheight)

    coverages = zeros(measured_emittances.size)
    emittances = zeros(measured_emittances.size)
    emittances_corrected = zeros(measured_emittances.size)
    for i in range(pepperpot_pitchs.size):
        coverage = pepperpot_pitchs[i]*number_of_holess / rms_beam_size_sample
        e = measured_emittances[0, i] / expected_e
        e_c = measured_emittances_corrected[0, i] / expected_e

        start = i*number_of_holess.size
        stop = (i+1)*number_of_holess.size

        coverages[start:stop] = coverage
        emittances[start:stop] = e
        emittances_corrected[start:stop] = e_c

    plt.figure('thesis', figsize=figsize)
    plt.plot(coverages, emittances, color=colours[0])
    plt.plot(coverages, emittances_corrected, color=colours[3])

    plt.xlim((0, coverages.max()))
    plt.xlabel('Pepperpot␣Extent␣($\sigma$)')
    plt.ylabel('Accuracy')

    plt.figure('thesis␣2', figsize=figsize)
    xs = pepperpot_pitchs[0]*number_of_holess
    ys = measured_emittances[0,0]
    ys2 = measured_emittances_corrected[0,0]
    plt.plot(xs*1e3, ys*1e9, color=colours[0])
    plt.plot(xs*1e3, ys2*1e9, color=colours[3])

    plt.axhline(expected_e*1e9, color=colours[1], ls=':')
    plt.axvline(rms_beam_size_sample*1e3, color='k', ls=':')

    plt.ylim((0, 150))

    plt.xlabel('Pepperpot␣Extent␣(mm)')
    plt.ylabel('Emittance␣(nm␣rad)')

    plt.tight_layout()

    plt.savefig('PepperpotExtent.pgf')
```

```
plt.show()
```

B.2 Two-Dimensional Quadrupole and Pepperpot Simulations

This code was used in Sections 2.4 and 5.4. This code simulates charged particle beams, performing particle tracing, magnetic field interactions and optional particle-particle interactions. ElectronLens.py contains the class ElectronBunch which performs the beam simulation. MagneticFields.py contains a number of functions for the construction of magnetic fields for use with the simulation. EmittanceSim.py simulates the emittance measurements for Section 5.4 but the code shown in Section B.1 was preferable simply due to the time taken to perform equivalent simulations.

B.2.1 ElectronLens.py

```
##### Import #####
from numpy import sqrt, linspace, zeros, nansum, errstate, \
                  isfinite, absolute, arange, array, meshgrid, \
                  arctan2, cos, sin
from numpy.random import randn
from matplotlib import pyplot as plt, cm
from scipy.constants import pi, h, m_e, e, epsilon_0
from warnings import catch_warnings, filterwarnings
from multiprocessing import cpu_count, Pool
from functools import partial
from h5py import File

from MagneticFields import quadrupole, octupole, quadrupole2

##### Constants #####
coulomb_factor = 4*pi*epsilon_0 / e**2

##### Objects #####
class ElectronBunch:
    def __init__(self, n, energy, radius_x=1, radius_y=1, emittance=None,
                 blank=False):
        if not blank:
            self.setEnergy(energy)

            self.electrons = randn(n, 4)

            self.electrons[:, 0] = self.electrons[:, 0] * radius_x
            self.electrons[:, 1] = self.electrons[:, 1] * radius_y

            if emittance is None:
                self.electrons[:, 2:] = zeros((n, 2))
            else:
                self._setEmittance(emittance)

    def copy(self):
        new_bunch = ElectronBunch(0, 0)

        new_bunch.electrons = self.electrons.copy()
        new_bunch.setEnergy(self.getEnergy())

        return new_bunch

    def __iter__(self):
        return self.electrons.__iter__()

    def next(self):
        self.electrons.next()

    def getXs(self):
        return self.electrons[:, 0]

    def getYs(self):
        return self.electrons[:, 1]
```

```
def getVXs(self):
    return self.electrons[:, 2]

def getVYs(self):
    return self.electrons[:, 3]

def getXPrimes(self):
    return self.getVXs() / self.getSpeed()

def getYPrimes(self):
    return self.getVYs() / self.getSpeed()

def setEnergy(self, energy):
    self.energy = energy

def getEnergy(self):
    return self.energy

def getSpeed(self):
    return sqrt(2*self.energy/m_e)

def getWavelength(self):
    return h/(m_e*self.getSpeed())

def getCentre(self):
    return self.getXs().mean(), self.getYs().mean()

def getBunchWidth(self):
    return 2 * self.getXs().std(), 2 * self.getYs().std()

def getEmittance(self):
    """Get the RMS Emittance of the bunch."""
    x = self.getXs()
    y = self.getYs()

    xp = self.getXPrimes()
    yp = self.getYPrimes()

    rms_emittance_x = sqrt((x**2).mean()*(xp**2).mean() - (x*xp).mean()**2)
    rms_emittance_y = sqrt((y**2).mean()*(yp**2).mean() - (y*yp).mean()**2)

    return rms_emittance_x, rms_emittance_y

def setEmittance(self, emittance):
    self.electrons[:, 2:] = randn(self.electrons.shape[0], 2)

    self._setEmittance(emittance)

def _setEmittance(self, emittance):
    sigma_x, sigma_y = self.getXs().std(), self.getYs().std()

    # xp = x' = vx/vz
    # emittance = sqrt(det(sigma_matrix)) = sqrt(sig11*sig22-sig12**2)
    # sigma11=std_x**2, sigma22=std_y**2
    # sig12 = coupling between x and x'.
    # If we assume zero coupling then...
    sigma_xp = emittance/sigma_x
    sigma_yp = emittance/sigma_y

    # Assume that Vx, Vy are from the standard normal distribution.
    self.electrons[:, 2] *= sigma_xp * self.getSpeed()
    self.electrons[:, 3] *= sigma_yp * self.getSpeed()
    #   This assumes that emittance is geometric emittance and thus
    #   that sigma_p is from x' = dx/dz = v_x/v_z.

def propagate(self, dz, slices, B_field_func=None, multithread=False, z=0):
    dt = (dz/slices)/self.getSpeed()
    for _ in range(slices):
        new_electrons = zeros(self.electrons.shape)
        if multithread:
            func = partial(_propagate, speed=self.getSpeed(),
                           Xs=self.getXs(), Ys=self.getYs(),
```

```
                        dt=dt, B_field_func=B_field_func, z=z)

            with Pool(processes=cpu_count()) as workers:
                l = self.electrons.tolist()

                new_electrons = \
                    array(workers.map(func, l))

        else:
            for i, tron in enumerate(self.electrons):
                x, y, v_x, v_y = _propagate(tron, self.getSpeed(),
                                            self.getXs(), self.getYs(),
                                            dt, B_field_func, z)

                new_electrons[i, 0] = x
                new_electrons[i, 1] = y
                new_electrons[i, 2] = v_x
                new_electrons[i, 3] = v_y

        self.electrons[:, :] = new_electrons[:, :]

def plot(self, figname=None, color=None, quiver=False):
    plt.figure(figname)

    if quiver:
        plt.quiver(self.getXs(), self.getYs(),
                   self.getVXs(), self.getVYs(),
                   color=color)
    else:
        plt.plot(self.getXs(), self.getYs(), 'x', c=color)

    xlims = plt.xlim()
    ylims = plt.ylim()

    square = max(abs(xlims[0]), abs(xlims[1]),
                 abs(ylims[0]), abs(ylims[1]))

    plt.xlim((-square, square))
    plt.ylim((-square, square))

def plot_phasespace(self, figname=None, color=None):
    plt.figure(figname)

    # Plot X and Px
    plt.subplot(1, 2, 1)

    plt.title('X')
    plt.xlabel('X')
    plt.ylabel('Px')

    plt.plot(self.getXs(), self.getVXs()*m_e, 'x', c=color)

    # Plot Y and Py
    plt.subplot(1, 2, 2)

    plt.title('Y')
    plt.xlabel('Y')
    plt.ylabel('Py')

    plt.plot(self.getYs(), self.getVYs()*m_e, 'x', c=color)

def plot_distr_phase(self, figname=None, color=None):
    plt.figure(figname)

    # Plot the particle distribution.
    plt.subplot(2, 2, 1)

    plt.title('Electron␣Distribution')
    plt.xlabel('X')
    plt.ylabel('Y')

    plt.plot(self.getXs(), self.getYs(), ',')
```

```
        # Plot the Y phase space.
        plt.subplot(2, 2, 2)

        plt.title('Y␣Phase␣Space')
        plt.xlabel('Y')
        plt.ylabel('Py')

        plt.plot(self.getYs(), self.getVYs()*m_e, ',', c=color)

        # Plot the X phase space
        plt.subplot(2, 2, 3)

        plt.title('X␣Phase␣Space')
        plt.xlabel('X')
        plt.ylabel('Px')

        plt.plot(self.getXs(), self.getVXs()*m_e, ',', c=color)

    def __str__(self):
        return str(self.electrons)

##### Functions #####
def _null_B_field(x, y, z):
    return 0, 0, 0

def _propagate(electron, speed, Xs, Ys, dt, B_field_func, z):
    if B_field_func==None:
        B_field_func = _null_B_field

    x, y = electron[0], electron[1]
    v_x, v_y, v_z = electron[2], electron[3], speed
    B_x, B_y, B_z = B_field_func(electron[0], electron[1], z)

    if False:
        # Self interaction
        dxs = x - Xs
        dys = y - Ys

        denominator = sqrt(dxs**2+dys**2)**3 * coulomb_factor

        # When we get to the electron of interest we'll have a divide by zero.
        with catch_warnings():
            filterwarnings("ignore", category=RuntimeWarning)

            F_x = dxs / denominator
            F_y = dys / denominator

        F_x = F_x[isfinite(F_x)].sum()
        F_y = F_y[isfinite(F_y)].sum()
    else:
        F_x = 0
        F_y = 0

    F_x += -e*(v_y*B_z - v_z*B_y)
    F_y += -e*(v_z*B_x - v_x*B_z)

    dV_x = dt * F_x / m_e
    dV_y = dt * F_y / m_e

    dx = v_x*dt
    dy = v_y*dt

    return x + dx, y + dy, v_x + dV_x, v_y + dV_y

def set_bunch_speeds(trons, s_x=7277.6, s_y=7928.6):
    # Give them speeds so that they'll focus
    # Speed should be relative to r
    width_x, width_y = trons.getBunchWidth()

    for i in range(trons.electrons.shape[0]):
        electron = trons.electrons[i]

        r = sqrt(electron[0]**2 + electron[1]**2)
```

```
        r_factor = r/edge_of_bunch

        # Needs to be directed towards 0,0 which happens to be opposite to
        # the electron coordinates.
        electron[2] = -s_x * abs(electron[0]/width_x) * electron[0] / r
        electron[3] = -s_y * abs(electron[1]/width_y) * electron[1] / r

def set_noisey_bunch_speeds(trons, std_speed=10000, astigmatism=None):
    if astigmatism is None:
        trons.electrons[:, 2:] = randn(trons.electrons.shape[0], 2) * std_speed
    else:
        trons.electrons[:, 2] = randn(trons.electrons.shape[0]) * std_speed
        trons.electrons[:, 3] = randn(trons.electrons.shape[0]) * std_speed * astigmatism

def plot_field(field_func, radius, z=0, figname=None):
    n = 30
    field_factor = 4
    xs = linspace(-radius*field_factor, radius*field_factor, n)
    ys = linspace(-radius*field_factor, radius*field_factor, n)

    zs = array([0])

    field_x, field_y, field_z = \
        zeros((zs.size, ys.size, xs.size)), \
        zeros((zs.size, ys.size, xs.size)), \
        zeros((zs.size, ys.size, xs.size))
    for z in range(zs.size):
        for y in range(ys.size):
            for x in range(xs.size):



                B_x, B_y, B_z = field_func(xs[x], ys[y], zs[z])

                field_x[z, y, x] = B_x
                field_y[z, y, x] = B_y
                field_z[z, y, x] = B_z

    total_magnitude = sqrt(field_x**2 + field_y**2, field_z**2)

    # Plot transverse cross-section
    z_zero = absolute(zs - 0).argmin()

    horizontal = field_x[z_zero,:,:]
    vertical = field_y[z_zero,:,:]

    if figname==None:
        plt.figure('z=' + str(z))
    else:
        plt.figure(figname)

    magnitude = sqrt(horizontal**2 + vertical**2)
    plt.quiver(xs, ys,
        horizontal/magnitude,
        vertical/magnitude,
        magnitude,
        cmap='inferno')
    plt.xlim((xs.min()*(n+1)/n, xs.max()*(n+1)/n))
    plt.ylim((ys.min()*(n+1)/n, ys.max()*(n+1)/n))
    plt.xlabel('x')
    plt.ylabel('y')

    plt.gca().add_artist(plt.Circle((0, 0), radius, fill=False, color='r'))

    plt.axes().set_aspect('equal', 'box')

def simple_lens(bunch, strength, second_order=0, forth_order=0, sqrt_order=0):
    rs = sqrt(bunch.getXs()**2 + bunch.getYs()**2)
    angles = arctan2(bunch.getYs(), bunch.getXs())

    affects = strength * rs + second_order*rs**2 + forth_order*rs**4
```

```
    bunch.electrons[:, 2] += affects * cos(angles)
    bunch.electrons[:, 3] += affects * sin(angles)

def set_dataset(hdf, key, data):
    if key in hdf:
        del hdf[key]

    return hdf.create_dataset(key, data=data)

##### Script #####
if __name__=='__main__':
    if False:
        # Looking at performance with different geometry.
        edge_of_bunch_x = 0.005
        edge_of_bunch_y = 0.005
        initial_trons = ElectronBunch(100, 17.1*e/2, edge_of_bunch_x, edge_of_bunch_y)

        set_noisey_bunch_speeds(initial_trons)

        min_stretch = 1
        max_stretch = 5
        n_stretches = 5
        stretches = array([0.1, 5])#linspace(min_stretch, max_stretch, n_stretches)

        min_width = 0.05
        max_width = 0.3
        n_widths = 1
        widths = array([0.015])#linspace(min_width, max_width, n_widths)

        min_current = 1
        max_current = 100
        n_currents = 5
        currents = linspace(min_current, max_current, n_currents)

        B_start = 0.2

        min_z = 0 - B_start
        max_z = 1 - B_start
        n_zs = 100
        zs = linspace(min_z, max_z, n_zs)

        data_stretch = []
        count = 1
        for m, stretch in enumerate(stretches):
            data = []
            for k, width in enumerate(widths):
                data_width = []

                current_mins = zeros(n_currents)
                current_mins_current = zeros(n_currents)
                for j, current in enumerate(currents):
                    trons = initial_trons.copy()

                    # Propagate through the octupole lens.
                    octupole_radius = 0.08


                    solenoid_transverse_radius = width*stretch
                    solenoid_longitudinal_radius = width
                    B_func = quadrupole2(current=current, inner_radius=octupole_radius,
                        solenoid_transverse_radius=solenoid_transverse_radius,
                        solenoid_longitudinal_radius=solenoid_longitudinal_radius,
                        solenoid_turns=1, solenoid_length=.01,
                        angle=pi/4)


                    B_f = lambda x, y, z: B_func(x, y, z) \
                        if abs(z)<width*4 else (0, 0, 0)

                    #plot_field(B_func, octupole_radius, z=B_start)

                    width_xs = zeros(n_zs)
                    width_ys = zeros(n_zs)
```

```
                for i, z in enumerate(zs):
                    print('{:} of {:}'.format(
                        count, n_stretches*n_widths*n_currents*n_zs))
                    print('\t\t', solenoid_transverse_radius)
                    count += 1

                    trons.propagate(zs[1]-zs[0], 2, multithread=False,
                        B_field_func=B_f, z=z)

                    w_x, w_y = trons.getBunchWidth()

                    width_xs[i] = w_x
                    width_ys[i] = w_y

                data_width.append((width_xs, width_ys))

            data.append(data_width)

        #plot_field(B_func, octupole_radius, z=B_start, figname=str(stretch))
        data_stretch.append(data)

    with File('electron lens datam mk2.h5') as hdf:
        for k, stretch in enumerate(stretches):
            stretch_group = hdf.require_group('s ' + str(stretch))
            for i, width in enumerate(widths):
                width_group = stretch_group.require_group('w ' + str(width))
                for j, current in enumerate(currents):
                    current_group = width_group.require_group('I ' + str(current))

                    name = 'W ' + str(width) + ' I ' + str(current)

                    z_group = current_group.require_group(name)

                    if 'x' in z_group:
                        del z_group['x']
                    if 'y' in z_group:
                        del z_group['y']
                    if 'z' in z_group:
                        del z_group['z']

                    z_group.create_dataset('x', data=data[i][j][0])
                    z_group.create_dataset('y', data=data[i][j][1])
                    z_group.create_dataset('z', data=zs)
else:
    # Take 2
    zs_key = 'zs'
    currents_key = 'quadrupole currents'
    logitudinal_width_key = 'longitudinal width'
    stretch_key = 'stretch'
    widths_key = 'beam widths'

    #hdf_name =AR 'QuadrupoleSims.h5'
    #hdf_name = 'QuadrupoleSims-Long2.h5'
    #hdf_name = 'QuadrupoleSims-Long3.h5'
    #hdf_name = 'QuadrupoleSims-Long4.h5'
    hdf_name = 'QuadrupoleSims-Long5.h5'
    with File(hdf_name) as hdf:
        if False:
            # RUN THE SIM

            # Create bunch
            edge_of_bunch_x = 0.005
            edge_of_bunch_y = 0.005
            initial_trons = ElectronBunch(100, 17.1*e/2, edge_of_bunch_x, edge_of_bunch_y)

            # Make the bunch noisey and astigmatic. (1 = no astigmatism)
            set_noisey_bunch_speeds(initial_trons, astigmatism=2)

            # Z axis
            min_z = 0
            max_z = 1
            n_zs = 2000
            zs = linspace(min_z, max_z, n_zs)
```

```
# Focusing lens
lens_strength = -6e6
focusing_lens_z = (min_z+max_z)/2
focused = False

# QUADRUPOLE LENS PARAMETERS
quad_lens_z = (min_z + focusing_lens_z) /2

quadrupole_radius = 0.045 # radius from centre of electron beam to solenoids

# Current through solenoids.
min_current = 10
max_current = 30
n_currents = 20
quad_currents = linspace(min_current, max_current, n_currents)

# Longitudinal width/depth of solenoids
min_width = 0.010
max_width = 0.010
n_widths = 1
solenoid_longitudinal_radiuss = array([0.015])#linspace(min_width, max_width, n_widths)

# Ellipticity of solenoids. Transverse width = depth * stretch
min_stretch = 1
max_stretch = 7
n_stretches = 14
stretchs = linspace(min_stretch, max_stretch, n_stretches)

# Widths of the beam
widths = zeros((quad_currents.size, solenoid_longitudinal_radiuss.size, \
                stretchs.size, zs.size, 2))

# HDF stuff.
set_dataset(hdf, zs_key, zs)
set_dataset(hdf, currents_key, quad_currents)
set_dataset(hdf, logitudinal_width_key, solenoid_longitudinal_radiuss)
set_dataset(hdf, stretch_key, stretchs)
widths = set_dataset(hdf, widths_key, widths)

# Simulate
number_of_calcs = quad_currents.size * solenoid_longitudinal_radiuss.size * \
                  stretchs.size * zs.size
count = 0
for i, quad_current in enumerate(quad_currents):
    for j, solenoid_longitudinal_radius in enumerate(solenoid_longitudinal_radiuss):
        for k, stretch in enumerate(stretchs):
            solenoid_transverse_radius = solenoid_longitudinal_radius * stretch

            # Quadrupole lens
            B_func = quadrupole2(current=quad_current, inner_radius=quadrupole_radius,
                            solenoid_transverse_radius=solenoid_transverse_radius,
                            solenoid_longitudinal_radius=solenoid_longitudinal_radius,
                            solenoid_turns=10, solenoid_length=.01,
                            angle=pi/4)

            # Only have quad field near the quadrupole
            B_f = lambda x, y, z: B_func(x, y, z) \
                if abs(z-quad_lens_z)<solenoid_longitudinal_radius*2 \
                        else (0, 0, 0)

            # Propagate the bunch along z.
            trons = initial_trons.copy()

            width_xs = zeros(n_zs)
            width_ys = zeros(n_zs)
            focused = False
            for m, z in enumerate(zs):
                print('{:} of {:}'.format(count, number_of_calcs))
                count += 1

                if not focused and z>focusing_lens_z:
                    simple_lens(trons, lens_strength)
```

```
                        focused = True

                    trons.propagate(zs[1]-zs[0], 2, multithread=False,
                        B_field_func=B_f, z=z)

                    w_x, w_y = trons.getBunchWidth()

                    width_xs[m] = w_x
                    width_ys[m] = w_y

                    widths[i, j, k, m, 0] = w_x
                    widths[i, j, k, m, 1] = w_y

                plt.figure('I:␣{:.1f}A,␣D:␣{:.1f}mm,␣W:␣{:.1f}mm'.format(quad_current, solenoid_longitudinal_radius*1e3, solen

                plt.plot(zs, width_xs)
                plt.plot(zs, width_ys)

                plt.axvline(focusing_lens_z, color='k', ls=':')
                plt.xlabel('z')
                plt.ylabel('widths')

else:
    # PLOT STUFF
    currents = array(hdf[currents_key])
    logitudinal_widths = array(hdf[logitudinal_width_key])
    stretchs = array(hdf[stretch_key])
    zs = array(hdf[zs_key])

    widths = array(hdf[widths_key])
    dz = zs[1]-zs[0]

    # For each stretch factor, find the current with the lowest astigmatism.
    # Lowest astigmatism is closest minima.
    cmap = cm.get_cmap('inferno')
    best_currents = zeros(stretchs.size, dtype='int')
    dzs = zeros(stretchs.size, dtype='int')
    waist_xs = zeros(stretchs.size)
    waist_ys = zeros(stretchs.size)
    for k, stretch in enumerate(stretchs):
        diffs = zeros(currents.size)
        for i, current in enumerate(currents):
            min_x_z = widths[i,0,k,:,0].argmin()
            min_y_z = widths[i,0,k,:,1].argmin()

            diffs[i] = min_x_z - min_y_z

        best_currents[k] = absolute(diffs).argmin()
        dzs[k] = absolute(diffs).min()

        #plt.figure('Stretch {:.1f}'.format(stretch))
        xs = widths[best_currents[k], 0, k, :, 0]
        ys = widths[best_currents[k], 0, k, :, 1]
        colour = cmap(k/stretchs.size)
        p = plt.plot(zs, xs, '-x', color=colour)
        plt.plot(zs, ys, '-x', color=p[-1].get_color())

        plt.axvline(zs[xs.argmin()], color=p[-1].get_color())
        plt.axvline(zs[ys.argmin()], color=p[-1].get_color())

        waist_x = xs.min()
        waist_y = ys.min()

        waist_x_z = zs[xs.argmin()]
        waist_y_z = zs[ys.argmin()]

        waist_xs[k] = waist_x
        waist_ys[k] = waist_y

        print('Factor:', stretch)
        print("Waist␣X:␣{:.2f}mm".format(1e3*waist_x))
        print("Waist␣Y:␣{:.2f}mm".format(1e3*waist_y))
```

```
                print("Waist X, z: {:.2f}mm".format(1e3*waist_x_z))
                print("Waist Y, z: {:.2f}mm".format(1e3*waist_y_z))
                print()

            p = plt.plot(zs, widths[0, 0, 0, :, 0], '-o')
            plt.plot(zs, widths[0, 0, 0, :, 1], '-o', color=p[-1].get_color())

            width = logitudinal_widths[0]

            from Thesis import thesis_init, linewidth, colours

            thesis_init()

            figwidth = linewidth
            figheight = linewidth/3

            plt.figure(figsize=(figwidth, figheight))
            num_rows = 1
            num_cols = 2

            plt.subplot(num_rows, num_cols, 1)
            #plt.title('Waist Separation')
            plt.plot(stretchs*width*1e3, 1e3*dzs*dz, 'x-', color=colours[0])
            plt.xlabel('Solenoid Transverse Radius (mm)')
            plt.ylabel('Waist Separation (mm)')

            plt.subplot(num_rows, num_cols, 2)
            #plt.title('Current Turns')
            plt.plot(stretchs*width*1e3, best_currents, 'x-', color=colours[0])
            plt.xlabel('Solenoid Transverse Radius (mm)')
            plt.ylabel('Current Turns (A turns)')

            if False:
                plt.subplot(num_rows, num_cols, 3)
                plt.title('Beam Waist')
                plt.plot(stretchs*width*1e3, waist_xs*1e3, 'x-', label='x')
                plt.plot(stretchs*width*1e3, waist_ys*1e3, 'x-', label='y')

                plt.xlabel('Solenoid Transverse Radius (mm)')
                plt.ylabel('Beam Waist (mm)')

            plt.tight_layout()

            plt.savefig('quad_sims.pgf')


    plt.show()
```

B.2.2 MagneticFields.py

```
##### Imports #####
from numpy import sqrt, sin, cos, arctan2, arange, meshgrid, linspace, degrees, zeros, absolute, array, matrix
from matplotlib import pyplot as plt
from matplotlib.colors import LogNorm
from scipy.constants import mu_0, pi
from scipy.special import agm
from scipy.integrate import quad
from functools import partial

##### Functions #####
def wire(current, loc_x, loc_y, plot=False, color='r'):
    if plot:
        plt.plot(loc_x, loc_y, 'o' if current>0 else 'x', color=color)

    def func(x, y):
        B_tangential = mu_0*current / (2*pi*sqrt((x-loc_x)**2 + (y-loc_y)**2))

        B_x = -B_tangential*sin(arctan2(y-loc_y, x-loc_x))
        B_y = B_tangential*cos(arctan2(y-loc_y, x-loc_x))

        return B_x, B_y, 0

    return func
```

```
def solenoid(current, loc_x, loc_y, angle, width, length, turns, plot=False, color='r'):
    locs = zeros((turns*2, 2))

    x_positions = linspace(-length/2, length/2, turns) if turns>1 else [0]
    for i, ex in enumerate(x_positions):
        wy = width/2

        locs[i][0] = ex
        locs[i][1] = wy

        wy = -width/2

        locs[i+turns][0] = ex
        locs[i+turns][1] = wy


    for i, (ex, wy) in enumerate(locs):
        locs[i][0] = ex*cos(angle) - wy*sin(angle) + loc_x
        locs[i][1] = ex*sin(angle) + wy*cos(angle) + loc_y

    funcs = []
    for i in range(turns):
        funcs.append(wire(current, locs[i][0], locs[i][1], plot, color=color))
        funcs.append(wire(-current, locs[i+turns][0], locs[i+turns][1], plot, color=color))


    def func(x, y):
        return map(sum, zip(*[f(x, y) for f in funcs]))

    return func

def quadrupole(current, loc_x, loc_y, radius, turns, width=None, angle=0, astig=1, plot=False, color='r', octupole=False):
    # astig is x current:y current, so astig 2 give twice the current in x
    # x, y, angle, current

    # If octupole==True then we do not flip the current for half the coils since this
    # function is being used to construct an octupole.
    current2 = current if octupole else -current

    locs = [[radius*cos(angle+0*pi/2)+loc_x,          radius*sin(angle+0*pi/2)+loc_y,
                    0 + angle, current],
            [radius*cos(angle+1*pi/2)+loc_x,          radius*sin(angle+1*pi/2)+loc_y,
                    2*pi/4 + angle, current2],
            [radius*cos(angle+2*pi/2)+loc_x,          radius*sin(angle+2*pi/2)+loc_y,
                    4*pi/4 + angle, current],
            [radius*cos(angle+3*pi/2)+loc_x,          radius*sin(angle+3*pi/2)+loc_y,
                    6*pi/4 + angle, current2],]

    if width==None:
        width = 2*pi*radius/8

    funcs = []
    for x, y, angle, current in locs:
        funcs.append(solenoid(current, x, y, angle, width, radius/4, turns, plot=plot, color=color))

    def func(x, y, z):
        return map(sum, zip(*[f(x, y) for f in funcs]))

    return func

def octupole(current, loc_x, loc_y, radius, turns, width=None, angle=0, astig=1, plot=False, color='r'):
    funcs = [
            quadrupole(current, loc_x, loc_y, radius, turns,
                width=width, angle=angle, astig=astig, plot=plot,
                color=color, octupole=True),
            quadrupole(-current, loc_x, loc_y, radius, turns,
                width=width, angle=angle+pi/4, astig=astig, plot=plot,
                color=color, octupole=True)
            ]

    def func(x, y):
        return map(sum, zip(*[f(x, y) for f in funcs]))
```

```
    return func

def circular_current_loop(current, radius, x0=0, y0=0, z0=0,
                  thetaX=0, thetaY=0, thetaZ=0, rotation_matrix=None):
    return elliptical_current_loop(current, radius, radius,
                                   x0=x0, y0=y0, z0=z0,
                                   thetaX=thetaX, thetaY=thetaY, thetaZ=thetaZ,
                                   rotation_matrix=rotation_matrix)

def elliptical_current_loop(current, radius_1, radius_2, x0=0, y0=0, z0=0,
                            thetaX=0, thetaY=0, thetaZ=0, rotation_matrix=None):
    # Inspired by http://www.physicspages.com/2013/04/18/magnetic-field-of-current-loop-off-axis-field/
    if rotation_matrix is None:
        Rx = matrix([[1,           0,            0],
                     [0, cos(thetaX), -sin(thetaX)],
                     [0, sin(thetaX),  cos(thetaX)]])

        Ry = matrix([[ cos(thetaY), 0, sin(thetaY)],
                     [           0, 1,           0],
                     [-sin(thetaY), 0, cos(thetaY)]])

        Rz = matrix([[cos(thetaZ), -sin(thetaZ), 0],
                     [sin(thetaZ),  cos(thetaZ), 0],
                     [          0,            0, 1]])

        rotation_matrix = Rx * Ry * Rz

    def func(x, y, z):
        # Centred at x0, y0, z0
        ex = x - x0
        wy = y - y0
        zd = z - z0

        ex, wy, zd = (rotation_matrix * matrix([ex, wy, zd]).T).flat

        # Current loop located at 0, 0, 0 in the x, y, plane.

        denominator = lambda phi: ( ex**2 + wy**2 + zd**2  \
                                   +radius_1**2*cos(phi)**2\
                                   +radius_2**2*sin(phi)**2\
                                   -2*ex*radius_1*cos(phi) \
                                   -2*wy*radius_2*sin(phi) )**1.5

        f_x = lambda phi: zd*radius_2*cos(phi) / denominator(phi)
        f_y = lambda phi: zd*radius_1*sin(phi) / denominator(phi)
        f_z = lambda phi: ( radius_1*radius_2 \
                           -ex*radius_2*cos(phi) - wy*radius_1*sin(phi) ) \
                                                 / denominator(phi)

        B_x = mu_0*current / (4*pi) * quad(f_x, 0, 2*pi)[0]
        B_y = mu_0*current / (4*pi) * quad(f_y, 0, 2*pi)[0]
        B_z = mu_0*current / (4*pi) * quad(f_z, 0, 2*pi)[0]

        # Undo rotation
        B_x, B_y, B_z = (rotation_matrix.I*matrix([B_x, B_y, B_z]).T).flat

        return B_x, B_y, B_z

    return func

def current_loop(current, radius,
                 x0=0, y0=0, z0=0,
                 thetaX=0, thetaY=0, thetaZ=0, rotation_matrix=None):
    # Inspired by http://www.physicspages.com/2013/04/18/magnetic-field-of-current-loop-off-axis-field/
    if rotation_matrix is None:
        Rx = matrix([[1,           0,            0],
                     [0, cos(thetaX), -sin(thetaX)],
                     [0, sin(thetaX),  cos(thetaX)]])

        Ry = matrix([[ cos(thetaY), 0, sin(thetaY)],
                     [           0, 1,           0],
                     [-sin(thetaY), 0, cos(thetaY)]])
```

```
        Rz = matrix([[cos(thetaZ), -sin(thetaZ), 0],
                     [sin(thetaZ),  cos(thetaZ), 0],
                     [          0,            0, 1]])

        rotation_matrix = Rx * Ry * Rz

    def func(x, y, z):
        # Centred at x0, y0, z0
        ex = x - x0
        wy = y - y0
        zd = z - z0

        ex, wy, zd = (rotation_matrix * matrix([ex, wy, zd]).T).flat

        # Current loop located at 0, 0, 0 in the x, y, plane.
        denominator = lambda phi: ( radius**2 + \
                                    ex**2 + wy**2 + zd**2 \
                                    - 2*wy*radius*sin(phi) \
                                    - 2*ex*radius*cos(phi) )**1.5

        f_x = lambda phi: zd*cos(phi) / denominator(phi)
        f_y = lambda phi: zd*sin(phi) / denominator(phi)
        f_z = lambda phi: (radius - ex*cos(phi) - wy*sin(phi)) / denominator(phi)

        B_x = mu_0*current*radius / (4*pi) * quad(f_x, 0, 2*pi)[0]
        B_y = mu_0*current*radius / (4*pi) * quad(f_y, 0, 2*pi)[0]
        B_z = mu_0*current*radius / (4*pi) * quad(f_z, 0, 2*pi)[0]

        # Undo rotation
        B_x, B_y, B_z = (rotation_matrix.I*matrix([B_x, B_y, B_z]).T).flat

        return B_x, B_y, B_z

    return func

def solenoid2(current, inner_radius_1, inner_radius_2, length, turns_per_layer,
                  x0=0, y0=0, z0=0,
                  thetaX=0, thetaY=0, thetaZ=0, rotation_matrix=None):
    if rotation_matrix is None:
        Rx = matrix([[1,           0,            0],
                     [0, cos(thetaX), -sin(thetaX)],
                     [0, sin(thetaX),  cos(thetaX)]])

        Ry = matrix([[ cos(thetaY), 0, sin(thetaY)],
                     [           0, 1,           0],
                     [-sin(thetaY), 0, cos(thetaY)]])

        Rz = matrix([[cos(thetaZ), -sin(thetaZ), 0],
                     [sin(thetaZ),  cos(thetaZ), 0],
                     [          0,            0, 1]])

        rotation_matrix = Rz * Rx * Ry

    wire_diameter = length/(turns_per_layer)
    funcs = []

    for turn in range(turns_per_layer):
        offset = ((turn+0.5)/turns_per_layer - 0.5)*length

        offset_x, offset_y, offset_z = (rotation_matrix * matrix([0, 0, offset]).T).flat

        ex0 = x0 + offset_x
        wy0 = y0 + offset_y
        zd0 = z0 + offset_z

        solenoid_transverse_radius = inner_radius_1
        solenoid_longitudinal_radius = inner_radius_2

        loop_func = elliptical_current_loop(current,
                    solenoid_longitudinal_radius, solenoid_transverse_radius,
                    x0=ex0, y0=wy0, z0=zd0,
                    thetaX=thetaX, thetaY=thetaY, thetaZ=thetaZ)
```

```
        funcs.append(loop_func)

    def func(x, y, z):
        return map(sum, zip(*[f(x, y, z) for f in funcs]))

    return func

def quadrupole2(current, inner_radius,
                solenoid_transverse_radius, solenoid_longitudinal_radius,
                solenoid_turns, solenoid_length,
                x0=0, y0=0, z0=0, angle=0):
    print('Ocutpole␣Radius:', inner_radius,
        '\nSolenoid␣(transverse,␣long):', solenoid_transverse_radius, solenoid_longitudinal_radius,
        '\nSolenoid␣(turns,␣length):', solenoid_turns, solenoid_length,
        '\nx,␣y,␣z:', x0, y0, z0)
    funcs = []

    if solenoid_turns==1:
        R = inner_radius
    else:
        R = inner_radius+solenoid_length/2

    for i in range(4):
        x = R*cos(i*pi/2 + angle) - x0
        y = R*sin(i*pi/2 + angle) - y0
        z = -z0

        I = current #if i%2==0 else -current
        funcs.append(solenoid2(I,
                        solenoid_transverse_radius, solenoid_longitudinal_radius,
                        solenoid_length, solenoid_turns,
                        x, y, z, 0, pi/2, i*pi/2 - angle))

    def func(x, y, z):
        return map(sum, zip(*[f(x, y, z) for f in funcs]))

    return func

##### Script #####
if __name__ == '__main__':
    n = 20
    xs = linspace(-10, 10, n)
    ys = linspace(-10, 10, n)
    #xs = linspace(-0.08*4, 0.08*4, n)
    #ys = linspace(-0.08*4, 0.08*4, n)
    zs = array([0.2])#linspace(-10, 10, n)

    xx, yy = meshgrid(xs, ys, sparse=True)

    if True:
        # 2D
        field_x, field_y = zeros((ys.size, xs.size)), zeros((ys.size, xs.size))

        if False:
            field_x, field_y = wire(2, 0, 0, True)(xx, yy)

        if False:
            for ex in range(-8, 9, 2):
                for wy in [-2, 2]:
                    f_x, f_y = wire(wy, ex, wy, True)(xx, yy)

                    field_x += f_x
                    field_y += f_y

        if False:
            field_x, field_y = solenoid(3, 0, 0, 0, 3, 3, 10, True)(xx, yy)

        if False:
            f_x, f_y = solenoid(3, 5, 0, 0, 3, 3, 10, True)(xx, yy)
            field_x += f_x
            field_y += f_y
```

```
    f_x, f_y = solenoid(-3, 0, 5, pi/2, 3, 3, 10, True)(xx, yy)
    field_x += f_x
    field_y += f_y

    f_x, f_y = solenoid(3, -5, 0, pi, 3, 3, 10, True)(xx, yy)
    field_x += f_x
    field_y += f_y

    f_x, f_y = solenoid(-3, 0, -5, 3*pi/2, 3, 3, 10, True)(xx, yy)
    field_x += f_x
    field_y += f_y

if False:
    field_x, field_y = octupole(1, 0, 0, 5, 1, width=2, angle=pi/8, astig=1, plot=True)(xx, yy)

if True:
    # For Thesis
    green = (77/255, 155/255, 77/255)
    linewidth = 5.71
    figsize=(linewidth/2, linewidth/2)
    plt.figure(1, figsize=figsize)

    current = 10
    radius = 7.5
    n_turns = 3
    coil_width = 3.5

    field_x, field_y, field_z = quadrupole(current, 0, 0, radius, n_turns, width=coil_width, astig=0.8, angle=pi/4, plot=True, colo

    # And again to get solenoids on second subplot.
    plt.figure(2, figsize=figsize)
    quadrupole(current, 0, 0, radius, n_turns, width=coil_width, astig=0.8, angle=pi/4, plot=True, color=green)

if False:
    field_x2, field_y2 = octupole(50, 0, 0, 8, 1, width = 0.01, astig=1, plot=True)(xx, yy)

    plt.quiver(xs, ys,
        field_x2/sqrt(field_x2**2 + field_y2**2),
        field_y2/sqrt(field_x2**2 + field_y2**2),
        sqrt(field_x2**2 + field_y2**2),
        cmap='inferno')
    plt.xlim((xs.min(), xs.max()))
    plt.ylim((ys.min(), ys.max()))

    plt.figure()

    field_x, field_y = octupole(50, 0, 0, 8, 1, width = None, astig=1, plot=True)(xx, yy)

if False:
    field_x, field_y, field_z = \
        zeros((ys.size, xs.size)), zeros((ys.size, xs.size)),zeros((ys.size, xs.size))

    for i in range(ys.size):
        for j in range(xs.size):
            for k in range(zs.size):
                B_x, B_y, B_z =\
                    solenoid2(xs[j], ys[i], zs[k], radius=1, length=1, current=1, x_loc=0, y_loc=0, z_loc=0)

                field_x[i, j] = B_x
                field_y[i, j] = B_y
                field_z[i, j] = B_z


# For Thesis Plot
v_z = 1
force_x = -v_z*field_y
force_y = v_z*field_x

plt.figure(1, figsize=figsize)
plt.quiver(xs, ys,
    field_x/sqrt(field_x**2 + field_y**2),
    field_y/sqrt(field_x**2 + field_y**2),
    sqrt(field_x**2 + field_y**2),
```

```
        cmap='inferno')
    plt.xlim((xs.min(), xs.max()))
    plt.ylim((ys.min(), ys.max()))

    plt.xticks([])
    plt.yticks([])

    plt.tight_layout()

    plt.savefig('QuadrupoleField.pgf')

    plt.figure(2, figsize=figsize)
    plt.quiver(xs, ys,
        force_x/sqrt(force_x**2 + force_y**2),
        force_y/sqrt(force_x**2 + force_y**2),
        sqrt(field_x**2 + force_y**2),
        cmap='inferno')

    plt.xticks([])
    plt.yticks([])

    plt.xlim((xs.min(), xs.max()))
    plt.ylim((ys.min(), ys.max()))

    plt.tight_layout()

    plt.savefig('QuadrupoleForce.pgf')


elif True:
    # 3D
    field_x, field_y, field_z = zeros((zs.size, ys.size, xs.size)),\
        zeros((zs.size, ys.size, xs.size)), zeros((zs.size, ys.size, xs.size))

    radius = 2.5
    length = 10
    current = 10
    if True:
        if False:
            func = current_loop(1, 5)
        elif False:
            func = solenoid2(-1, 5, 5, 3,
                            x0=0, y0=0, z0=0,
                            thetaX=0, thetaY=pi/2, thetaZ=pi/2)
        elif False:
            func = quadrupole2(current=1, inner_radius=5,
                    solenoid_transverse_radius=3, solenoid_longitudinal_radius=3,
                    solenoid_turns=1, solenoid_length=3,
                    angle=pi/4)

        elif True:
            func = quadrupole2(current=1, inner_radius=0.08,
                    solenoid_transverse_radius=0.01, solenoid_longitudinal_radius=0.01,
                    solenoid_turns=1, solenoid_length=3,
                    z0=0.2,
                    angle=pi/4)

        elif False:
            func = solenoid2(1, 1, 5, 5,
                    5, 0, 0, 0, pi/2, 0)

        field_x, field_y, field_z = \
            zeros((zs.size, ys.size, xs.size)), \
            zeros((zs.size, ys.size, xs.size)), \
            zeros((zs.size, ys.size, xs.size))
        for z in range(zs.size):
            for y in range(ys.size):
                for x in range(xs.size):



                    B_x, B_y, B_z = func(xs[x], ys[y], zs[z])
```

```
            field_x[z, y, x] = B_x
            field_y[z, y, x] = B_y
            field_z[z, y, x] = B_z

total_magnitude = sqrt(field_x**2 + field_y**2, field_z**2)

# Plot transverse cross-section
z_zero = absolute(zs - 0).argmin()

horizontal = field_x[z_zero,:,:]
vertical = field_y[z_zero,:,:]

plt.figure('Transverse')
magnitude = sqrt(horizontal**2 + vertical**2) + 1e-12
plt.quiver(xs, ys,
    horizontal/magnitude,
    vertical/magnitude,
    magnitude,
    cmap='inferno')
plt.xlim((xs.min()*(n+1)/n, xs.max()*(n+1)/n))
plt.ylim((ys.min()*(n+1)/n, ys.max()*(n+1)/n))
plt.xlabel('x')
plt.ylabel('y')

plt.gca().add_artist(plt.Circle((0, 0), radius, fill=False, color='r'))

plt.axes().set_aspect('equal', 'box')

if False:
    # Plot longitudinal cross-section
    # Bz vs By
    x_zero = absolute(xs - 0).argmin()

    horizontal = field_z[:,:,x_zero].T
    vertical = field_y[:,:,x_zero].T

    magnitude = sqrt(horizontal**2 + vertical**2)

    plt.figure('Longitudinal␣Y')
    plt.quiver(zs, ys,
        horizontal/magnitude,
        vertical/magnitude,
        magnitude,
        cmap='inferno')
    plt.xlim((zs.min()*(n+1)/n, zs.max()*(n+1)/n))
    plt.ylim((ys.min()*(n+1)/n, ys.max()*(n+1)/n))
    plt.xlabel('z')
    plt.ylabel('y')

    plt.gca().add_artist(plt.Rectangle((0-length/2, 0-radius),
                length, radius*2, fill=False, color='r'))

    plt.axes().set_aspect('equal', 'box')

    # Bz vs Bx
    y_zero = absolute(ys - 0).argmin()

    horizontal = field_z[:,y_zero,:].T
    vertical = field_x[:,y_zero,:].T

    magnitude = sqrt(horizontal**2 + vertical**2)

    plt.figure('Longitudinal␣X')
    plt.quiver(zs, xs,
        horizontal/magnitude,
        vertical/magnitude,
        magnitude,
        cmap='inferno')
    plt.xlim((zs.min()*(n+1)/n, zs.max()*(n+1)/n))
    plt.ylim((xs.min()*(n+1)/n, xs.max()*(n+1)/n))
    plt.xlabel('z')
    plt.ylabel('x')
```

```
                plt.gca().add_artist(plt.Rectangle((0-length/2, 0-radius),
                            length, radius*2, fill=False, color='r'))

                plt.axes().set_aspect('equal', 'box')
    plt.show()
```

B.2.3 EmittanceSim.py

```
# -*- coding: utf-8 -*-
"""
Created on Mon Jan 16 16:50:22 2017

@author: Joshua Torrance
"""

# Imports
from numpy import zeros, fromfunction, linspace, sqrt
from scipy.signal import fftconvolve
from scipy.constants import e
from matplotlib import pyplot as plt

from ElectronLens import ElectronBunch
from Gaussians import gaussian2d
from Fitting import fitCurve


# Constants
m_per_pixel = 0.04/(490*2)
single_e_spread = 4.795 * m_per_pixel


# Functions
def detectDistribution(bunch, spread_m=single_e_spread,
                       pixel_size_m=m_per_pixel):
    x_pos = bunch.getXs()
    y_pos = bunch.getYs()

    image_length_pix = int(max((x_pos.max() - x_pos.min()),
                           (y_pos.max() - y_pos.min())) / pixel_size_m) + 1

    image = zeros((image_length_pix, image_length_pix))

    for x, y in zip(x_pos, y_pos):
        i = int((x-x_pos.min())/pixel_size_m)
        j = int((y-y_pos.min())/pixel_size_m)

        image[i, j] = 1

    spread_length_pix = int(spread_m * 3 / pixel_size_m)

    def spread_func(i, j):
        return gaussian2d(1, spread_length_pix//2, spread_length_pix//2,
                          spread_length_pix, spread_length_pix, 0)(i, j)
    spread_image = fromfunction(spread_func,
                                (spread_length_pix, spread_length_pix))

    convolved_image = fftconvolve(image, spread_image, mode='same')

    return convolved_image


def calculate_emittance(zs, widths):
    zs -= zs[0]

    sigma_11s = widths**2
    sigma_0_11 = widths[0]**2

    def sigma_11_func(z, sigma_0_11, sigma_0_12, sigma_0_22):
        return sigma_0_11 + 2*z*sigma_0_12 + z**2*sigma_0_22

    def fit_func(z, sigma_0_12, sigma_0_22):
        return sigma_11_func(z, sigma_0_11, sigma_0_12, sigma_0_22)
```

```
    guess = 1, 1

    fit = fitCurve(zs, sigma_11s, fit_func, guess)
    fit_sigma_0_12, fit_sigma_0_22 = fit

    plt.plot(zs, sigma_11s, 'x')

    fit_zs = linspace(zs[0], zs[-1], 1000)
    fit_sigma_11s = sigma_11_func(fit_zs,
                                  sigma_0_11, fit_sigma_0_12, fit_sigma_0_22)

    plt.plot(fit_zs, fit_sigma_11s)

    emittance = sqrt(sigma_0_11*fit[-1]-fit[1]**2)

    return emittance

# Sciprt
if __name__ == '__main__':
    bunch_edge_x = 0.005
    bunch_edge_y = 0.005
    initial_emittance = 0.00001
    trons = ElectronBunch(10000, 17.1*e/2, bunch_edge_x, bunch_edge_y,
                          emittance=initial_emittance)

    zs = linspace(0, 2, 20)
    widths = zeros(zs.shape)
    widths[0] = trons.getXs().std()
    for i in range(zs.size-1):
        dz = zs[i+1] - zs[i]

        print(i, 'E␣=', trons.getEmittance())
        trons.propagate(dz, 10)

        widths[i+1] = trons.getXs().std()

    emittance = calculate_emittance(zs, widths)

    print('Set␣Emittance:', initial_emittance)
    print('True␣Emittance:', trons.getEmittance())
    print('Measured␣emittance:', emittance)

    plt.show()
```

B.3 Image Registration

This code was used to perform the image processing and registration for the diffraction measurements in Chapter 4 and the brightness measurements in Chapter 5.

B.3.1 ImageSet.py

```
###### Imports ######
from glob import glob
from os.path import isdir
from re import split
from multiprocessing import cpu_count, Pool
from functools import partial
from numpy import array, unravel_index, pad, zeros, round, floor, ceil
from PIL.Image import open
from scipy.signal import fftconvolve
from h5py import File

###### Cosntants ######
DEFAULT_HDF_NAME = 'Images.h5'
REGISTERED_NAME_SUFFIX = '␣(registered)'
PGM_EXTENTION = '.pgm'


###### Functions ######
# Go through directories process images sets.
# Assumes that when images are found there are no sub directories in that dir.
def processDirs(hdf_name=DEFAULT_HDF_NAME, directory='./', register=True, \
                multi_process=True, reg_iterations=1, chunk_size=0, \
                max_deviation=(None, None), image_extention=PGM_EXTENTION):
    hdf = File(hdf_name)

    processImageSetSet(directory, hdf, register=register, \
                       multi_process=multi_process, \
                       reg_iterations=reg_iterations, \
                       chunk_size=chunk_size, \
                       max_deviation=max_deviation, \
                       image_extention=image_extention)

    return hdf

def processImageSetSet(directory, hdf, register=True, multi_process=True, \
                       reg_iterations=1, chunk_size=0, max_deviation=(None, None), \
                       image_extention=PGM_EXTENTION):
    stuff = glob(directory + '/*')

    for thing in stuff:
        if isdir(thing):
            group = hdf.require_group(dirToName(thing))

            processImageSetSet(thing, group, register=register, \
                               multi_process=multi_process, \
                               reg_iterations=reg_iterations, \
                               chunk_size=chunk_size, \
                               max_deviation=max_deviation, \
                               image_extention=image_extention)
        elif PGM_EXTENTION in thing:
                # Thar be images here.
                files = glob(directory + '/*' + image_extention)#[0:100]
                name = dirToName(directory)

                processImageSet(files, hdf, name, register=register, \
                                multi_process=multi_process, \
                                reg_iterations=reg_iterations, \
                                chunk_size=chunk_size, \
                                max_deviation=max_deviation)

                break
        else:
```

```
            # Not something we need to process.
            pass

# Process Image Sets into HDFs
def processImageSet(files, hdf, name, register=True, multi_process=True, \
                    reg_iterations=1, chunk_size=0, max_deviation=(None, None)):
    fetched = fetchImageSet(files, register=register, multi_process=multi_process, \
                            reg_iterations=reg_iterations, chunk_size=chunk_size, \
                            max_deviation=max_deviation)

    if register:
        average_image, registered_image = fetched
    else:
        average_image = fetched

    if name in hdf:
        del hdf[name]
    hdf.create_dataset(name, data=average_image)

    if register:
        reg_name = name + REGISTERED_NAME_SUFFIX
        if reg_name in hdf:
            del hdf[reg_name]

        hdf.create_dataset(reg_name, data=registered_image)


# 'Fetching' Image Sets
def fetchImageSet(files, register=True, multi_process=True, \
                  reg_iterations=1, chunk_size=0, max_deviation=(None, None)):
    images = loadImageSet(files, multi_process=multi_process)

    average_image = images.mean(axis=0)

    if register:
        registered_image = registerImageSet(images, multi_process=multi_process, \
                                            iterations=reg_iterations, chunk_size=chunk_size, \
                                             max_deviation=max_deviation)

        return average_image, registered_image
    else:
        return average_image

# Loading Images.
def loadImageSet(files, multi_process=True):
    # Prepare the function for the workers.
    load = partial(loadImageI, files=files)

    if multi_process:
        with Pool(processes=cpu_count()) as workers:
            ims = array(workers.map(load, range(len(files))))
    else:
        image = load(0)

        ims = zeros((len(files), image.shape[0], image.shape[1]))

        ims[0, :, :] = image

        for i in range(1, len(files)):
            ims[i, :, :] = load(i)

    return ims

def loadImageI(i, files):
    print('\tLoading image', i+1, 'of', len(files))

    return loadImage(files[i])

def loadImage(file):
    return removeTimestamp(array(open(file), dtype='float64'))

def removeTimestamp(image, mode='nearest'):
    # Images from PointGrey FlyCap2 can have timestamps which are the first 4 pixels.
```

```
    if mode=='nearest':
        image[0][0] = image[1][0]
        image[0][1] = image[1][1]
        image[0][2] = image[1][2]
        image[0][3] = image[1][3]
    elif mode=='mean':
        value = image.mean()

        image[0][0] = value
        image[0][1] = value
        image[0][2] = value
        image[0][3] = value
    elif mode=='zero':
        image[0][0] = 0
        image[0][1] = 0
        image[0][2] = 0
        image[0][3] = 0
    else:
        raise ValueError('Invalid mode supplied to ImageSet.removeTimestamp (' + str(mode) + ').')

    return image

# Registering Images
def registerImageSet(images, multi_process=True, reference_image=None, \
                     roi_centre=None, roi_width=400, iterations=1, chunk_size=0, \
                     max_deviation=(None, None)):
    if chunk_size==0 or images.shape[0]<=chunk_size:
        # Use correlation to determine the common centre
        centre_xs, centre_ys = correlateImageSet(images, multi_process=multi_process, \
                                                 reference_image=reference_image, \
                                                 roi_centre=roi_centre, \
                                                 roi_width=roi_width, \
                                                 max_deviation=max_deviation)

        if False:
            from matplotlib import pyplot as plt
            plt.figure()
            plt.subplot(2, 1, 1)
            plt.plot(centre_xs, label='x')
            diffs = []
            for i in range(centre_xs.size-1):
                dif = centre_xs[i]-centre_xs[i+1]
                diffs.append(dif)
                if abs(dif)>=20:
                    plt.plot(i+1, centre_xs[i+1], 'rx')
            plt.plot(diffs)
            plt.legend()

            plt.subplot(2, 1, 2)
            plt.plot(centre_ys, label='y')
            diffs = []
            for i in range(centre_ys.size-1):
                dif = centre_ys[i]-centre_ys[i+1]
                diffs.append(dif)
                if abs(dif)>=20:
                    plt.plot(i+1, centre_ys[i+1], 'rx')
            plt.plot(diffs)
            plt.legend()
            plt.show()

            exit()

        # Align the common centres.
        aligned_image = alignImageSet(images, centre_xs, centre_ys, multi_process=multi_process)

        if iterations>1:
            # Use the aligned_image as the new reference_image.

            # Increase the aligned image to the original size.
            d_size_x = images[0].shape[1] - aligned_image.shape[1]
            d_size_y = images[0].shape[0] - aligned_image.shape[0]

            new_reference_image = pad(aligned_image,
```

```
                                ((d_size_x, d_size_x), (d_size_y, d_size_y)),
                                mode='edge')

            return registerImageSet(images, multi_process=multi_process, \
                                    reference_image=new_reference_image, \
                                    roi_centre=roi_centre, roi_width=roi_width, \
                                    iterations=iterations-1, \
                                    max_deviation=max_deviation)
        else:
            return aligned_image
    else:
        chunks = [images[i:i+chunk_size] for i in range(0, images.shape[0], chunk_size)]

        padded_aligned_chunks = zeros((len(chunks), images.shape[1], images.shape[2]))
        for i, chunk in enumerate(chunks):
            aligned_chunk = array(registerImageSet(chunk, \
                                        multi_process=multi_process, \
                                        reference_image=reference_image, \
                                        roi_centre=roi_centre, roi_width=roi_width, \
                                        iterations=iterations, chunk_size=chunk_size, \
                                        max_deviation=max_deviation))

            # Increase the chunk to the original size.
            d_size_x = images[0].shape[1] - aligned_chunk.shape[1]
            d_size_y = images[0].shape[0] - aligned_chunk.shape[0]

            pad_left, pad_right = int(ceil(d_size_x/2)), int(floor(d_size_x/2))
            pad_bottom, pad_top = int(ceil(d_size_y/2)), int(floor(d_size_y/2))

            padded_aligned_chunk = pad(aligned_chunk,
                                    ((pad_bottom, pad_top), \
                                     (pad_left, pad_right)),
                                    mode='edge')

            padded_aligned_chunks[i, :, :] = padded_aligned_chunk

        return registerImageSet(padded_aligned_chunks, multi_process=multi_process, \
                                reference_image=None, \
                                roi_centre=roi_centre, roi_width=roi_width, \
                                iterations=iterations, chunk_size=0, \
                                max_deviation=max_deviation)

def correlateImageSet(images, multi_process=True, reference_image=None, roi_centre=None, \
                      roi_width=50, max_deviation=(None, None)):
    if reference_image is None:
        #reference_image = images[0]
        #reference_image = images[0:10].mean(axis=0)
        reference_image = images.mean(axis=0)

    if roi_centre is None:
        roi_centre = getMaxCoordinates(reference_image)


    subimages = getSubimages(images, roi_centre[0], roi_centre[1], roi_width)
    #reference_subimage = getSubimage(reference_image, roi_centre[0], roi_centre[1], roi_width)

    p_correlate = partial(correlate, reference_image=reference_image)

    if max_deviation[0] is None and max_deviation[1] is None and multi_process:
        with Pool(processes=cpu_count()) as workers:
                centre_xs, centre_ys = zip(*workers.map(p_correlate, enumerate(subimages)))

        centre_xs = array(centre_xs)
        centre_ys = array(centre_ys)
    else:
        centre_xs = zeros(subimages.shape[0])
        centre_ys = zeros(subimages.shape[0])

        for i, subimage in enumerate(subimages):
            centre_x, centre_y = p_correlate((i, subimage), \
                                        previous_centre=(centre_xs[i-1], centre_ys[i-1]) if i>0 else (None, None),
                                        max_deviation=max_deviation)
```

```
            centre_xs[i] = centre_x
            centre_ys[i] = centre_y

    # Shift to the full image coordinates
    start_x, stop_x, start_y, stop_y = getStartStop(roi_centre[0], roi_centre[1], roi_width, images[0].shape)
    centre_xs = centre_xs + start_x
    centre_ys = centre_ys + start_y

    return centre_xs, centre_ys

def correlate(enumerated_image, reference_image, previous_centre=(None,None), max_deviation=(None,None)):
    i, image = enumerated_image
    print('\tRegistering image', i+1)

    correlation = fftconvolve(image, reference_image[::-1, ::-1], mode='same')

    if previous_centre[0] is None or max_deviation[0] is None:
        start_x, stop_x = 0, correlation.shape[1]
    else:
        start_x = max(0, int(previous_centre[0] - max_deviation[0]))
        stop_x = min(correlation.shape[1], int(previous_centre[0] + max_deviation[0]))

    if previous_centre[1] is None or max_deviation[1] is None:
        start_y, stop_y = 0, correlation.shape[0]
    else:
        start_y = max(0, int(previous_centre[1] - max_deviation[1]))
        stop_y = min(correlation.shape[0], int(previous_centre[1] + max_deviation[1]))

    correlation = correlation[start_y:stop_y, start_x:stop_x]

    max_y, max_x = unravel_index(correlation.argmax(), correlation.shape)

    return max_x + start_x, max_y + start_y

def alignImageSet(images, centre_xs, centre_ys, multi_process=True):
    mean_x = round(centre_xs.mean())
    mean_y = round(centre_ys.mean())

    dxs = centre_xs - mean_x
    dys = centre_ys - mean_y

    min_dx = dxs.min()
    max_dx = dxs.max()
    min_dy = dys.min()
    max_dy = dys.max()

    p_align_image = partial(align_image, min_dx=min_dx, max_dx=max_dx,
                min_dy=min_dy, max_dy=max_dy)

    if multi_process:
        with Pool(processes=cpu_count()) as workers:
            output = workers.map(p_align_image, enumerate(zip(images, dxs, dys)))
            aligned_average_image = array(output).mean(axis=0)
    else:
        aligned_average_image = p_align_image((0, (images[0], dxs[0], dys[0])))

        for i in range(1, images.shape[0]):
            aligned_average_image += p_align_image((i, (images[i], dxs[i], dys[i])))

        aligned_average_image /= images.shape[0]

    return aligned_average_image

def align_image(tuple, min_dx, max_dx, min_dy, max_dy):
    i, (image, dx, dy) = tuple

    print('\tAligning image', i+1)

    x1 = int(-min_dx + dx)
    x2 = int(image.shape[1]-max_dx + dx)
    y1 = int(-min_dy + dy)
    y2 = int(image.shape[0]-max_dy + dy)
```

```
    return image[y1:y2, x1:x2]

# Utility Functions
def dirToName(dir):
    return split('/|\\\\', dir)[-1]
        # Split around / or \ (while escaping \'s).

def getStartStop2(x, y, length_x, length_y, shape):
    start_x = max(0, int(x - length_x/2))
    stop_x = min(shape[1], int(x + length_x/2))
    start_y = max(0, int(y - length_y/2))
    stop_y = min(shape[0], int(y + length_y/2))

    return start_x, stop_x, start_y, stop_y

def getStartStop(x, y, length, shape):
    return getStartStop2(x, y, length, length, shape)

def getSubimage(image, x, y, length):
    start_x, stop_x, start_y, stop_y = getStartStop(x, y, length, image.shape)

    return image[start_y:stop_y, start_x:stop_x]

def getSubimages(images, x, y, length):
    start_x, stop_x, start_y, stop_y = getStartStop(x, y, length, images[0].shape)

    return images[:, start_y:stop_y, start_x:stop_x]

def getMaxCoordinates(image):
    max_y, max_x = unravel_index(image.argmax(), image.shape)

    return max_x, max_y


##### Testing #####
if __name__=='__main__':
    # Some imports.
    from matplotlib import pyplot as plt

    from ImageUtility import plot_image

    # The script
    processDirs(directory='Test␣Images')

    with File(DEFAULT_HDF_NAME) as hdf:
        for key in hdf:
            arr = array(hdf[key])

            plt.figure(key)
            plot_image(arr, new_fig=False)

    plt.show()
```

B.4 Utility Code

Emittance.py was used for both the brightness measurements and simulations for Chapter 5. Fitting.py was used by a number of the codes in this appendix to fit functions to data.

B.4.1 Emittance.py

```
##### Imports #####
from numpy import sqrt, pi, nanmean, histogram, zeros, average, arange, floor, \
                  errstate, linspace, exp, absolute, array
from matplotlib import pyplot as plt
from scipy.constants import physical_constants, speed_of_light, m_e, eV, k as kB
from scipy.stats import norm
from scipy.signal import fftconvolve

from TraceUtility import window_trace, normalise
from Fitting import fitCurve


##### Constants #####
# Physical Constants
Plank = physical_constants['Planck constant in eV s'][0]
Ry = physical_constants['Rydberg constant'][0] # Rydberg constant (per m)
electric_field_atomic_unit = physical_constants['atomic unit of electric field'][0]
                # Atomic unit of electric field (V/m)

# Experimental Constants
mcp_calibration = 34.27 # counts/electron
Rb_ionisation_energy=4.1771270
red_wavelength = 780.2414e-9
electric_field = 16e3/0.05 # (V / m)
m_per_pixel = 58.2e-6
mcp_calibration = 34.27 # counts/electron
propagation_distance = 0.475

colors = ['b', 'g', 'r', 'c', 'm', 'y', 'k', 'chartreuse']


##### Functions #####
def normal_cumulative_thingy(x):
    f = lambda x: 1-(1-norm.cdf(x))*2

    return f(x)

def normal_prop(x_lo, x_hi):
    return norm.cdf(x_hi) - norm.cdf(x_lo)

def tophat_f(width, centre=0):
    return lambda x: absolute(x-centre)<width/2

def gaussian(x, height=1, centre=0, width=1, offset=0):
    return height * exp( -(x-centre)**2 / (2*width**2)) + offset

def gaussian_f(std):
    return lambda x: exp(-x**2/(2*std**2))

def n_gaussians(x, *args):
    y = zeros(x.size)
    for ey in arange(len(args)/3):
        # 3 arguments, ignore offset
        i = int(ey)*3

        height = args[i]
        centre = args[i+1]
        width = args[i+2]

        wy = gaussian(x, height, centre, width)

        y += wy
```

```
    return y

# Analysis
def emittance_from_line(line, peaks, sum_peaks=False, diag=False, m_per_pixel=m_per_pixel, \
                        number_holes=7, pitch=200e-6, propagation_distance=propagation_distance, adjust_for_size=True, return_beam_size=Tru
                        aperture_size=50e-6, adjust_for_aperture_size=True, refine_parameters=False, zero=True, index=-1, subplots=True):
    xs = arange(line.size, dtype='f')

    if diag and subplots:
        plt.subplot(2, 2, 1)
    ret = get_gaussians_from_line(xs, line, peaks, sum_peaks=sum_peaks, diag=diag, zero=zero)

    if sum_peaks:
        heights, centres, widths, sums = ret
    else:
        heights, centres, widths = ret
        sums = None

    if refine_parameters:
        try:
            if diag and subplots:
                plt.subplot(2, 2, 2)
            heights, centres, widths = refine_gaussian_parameters(xs, line, heights, centres, widths, diag=diag and subplots)
        except RuntimeError:
            # Leave them as they are.
            print('\tFailed to refine parameters.')

    try:
        if diag and subplots:
            plt.subplot(2, 2, 3)
        rms_size, total_beam_count, center = calc_beam_size(heights, centres, widths, \
                                        sums=sums, pepperpot_pitch=pitch, return_centre=True, \
                                        diag=diag and subplots)
    except RuntimeError as e:
        # Failed Fits. :(
        print('Failed to find beam size.')
        print(e)

        if False:
            plt.figure()
            if sums is not None:
                plt.title('Sums')
                plt.plot(sums)
            else:
                plt.title('Heights')
                plt.plot(heights)
            plt.figure()
            plt.plot(line)

        em = float('NaN')
        total_beam_count = float('NaN')
        rms_size = float('NaN')

        #rms_size, total_beam_count = -1, 0
    else:
        em = emittance(heights, centres, widths, num_particles=sums, m_per_pixel=m_per_pixel, \
                       number_holes=number_holes, pitch=pitch, propagation_distance=propagation_distance,
                       aperture_size=aperture_size, adjust_for_aperture_size=adjust_for_aperture_size)

        if adjust_for_size and rms_size!=-1:
            if False:
                # Centred correction
                correction = normal_cumulative_thingy(0.5*number_holes*pitch/rms_size)
            else:
                # Off centred correction
                x_lo = (3/4)*(center - number_holes*pitch/2) / rms_size
                x_hi = (3/4)*(center + number_holes*pitch/2) / rms_size
                    # Why 3/4? 1 rms to either side = /2

                correction = normal_prop(x_lo, x_hi)

            print('\tBeam rms size: {:.2f}um'.format(rms_size*1e6))
```

```
            print('\tCorrection: {:.2f}'.format(correction))
            em /= correction
            #print('\t{:.2f}'.format(correction))
        else:
            print('\tNot correcting.')

    if return_beam_size and return_beam_count:
        return em, rms_size, total_beam_count
    elif return_beam_size and not return_beam_count:
        return em, rms_size
    elif not return_beam_size and return_beam_count:
        return em, total_beam_count
    else:
        return em

def get_gaussians_from_line(xs, ys, peaks, sum_peaks=False, diag=False, zero=True):
    if zero:
        hist, bin_edges = histogram(ys, bins=int(ys.size))
        c = bin_edges[hist.argmax()]

        ys -= c

    if diag:
        #plt.figure()
        plt.plot(xs, ys)

    heights = zeros(len(peaks))
    widths = zeros(len(peaks))
    centres = zeros(len(peaks))
    sums = zeros(len(peaks))
    for i in range(peaks.size):
        if i==0:
            lo = peaks[i] - (peaks[i+1] - peaks[i])/2
        else:
            lo = hi

        if i==peaks.size-1:
            hi = peaks[i] + (peaks[i] - peaks[i-1])/2
        else:
            hi = (peaks[i] + peaks[i+1])/2

        exs, wys = window_trace(xs, ys, lo, hi)

        # Mebbe?
        #wys2 = wys - wys.min()
        wys2 = wys

        middle = average(exs, weights=wys2)

        # Occasionally we get -'ve numbers in ys due to subtracting c.
        # Set these to 0 so the sqrt doesn't fail.
        wys2[wys2<0] = zeros((wys2<0).sum())
        if wys.sum()==0:
            # With zero particles the std shouldn't matter.
            # Setting to very small to avoid divide by zero later.
            std = 1e-12
        else:
            std = sqrt(average((exs-middle)**2, weights=wys2))

        max = wys.max()
        sum = wys.sum()

        heights[i] = max
        widths[i] = std
        centres[i] = middle
        sums[i] = sum
        if diag:
            colour = colors[(i+1) % len(colors)]
            plt.plot(exs, wys, color=colour)
            plt.axvline(middle, color=colour)
            plt.axvspan(middle-std/2, middle+std/2, color=colour, alpha=0.25)
            plt.text(10, 0.9*ys.max()*(heights.size-i)/heights.size,
                     "x={:.2f}\ns={:.2f}".format(middle, std), color=colour, va='top')
```

```
            if i==3:
                to_plot_middle = middle
                to_plot_xs = exs-middle
                to_plot_ys = wys

    if sum_peaks:
        return heights, centres, widths, sums
    else:
        return heights, centres, widths

def emittance(heights, centres, widths, number_holes=7, pitch=200e-6, \
              propagation_distance=propagation_distance, m_per_pixel=m_per_pixel, \
              num_particles=None, aperture_size=50e-6, adjust_for_aperture_size=True):
    if num_particles is None:
        num_particles = sqrt(2*pi) * heights * widths * m_per_pixel
        # Not actually number of particles. Number of camera counts.

    # Pepperpot hole locations.
    slit_positions = arange(0, heights.size * pitch * 0.999, pitch)
        # Occasionally returns one too many elements.
        # Probably due to binary accuracy. So: * 0.999

    # Take the middle hole as the 'centre'
    slit_positions -= slit_positions[int(floor(number_holes)/2)]
    mean_slit_position = average(slit_positions, weights=num_particles)

    middle = centres[int(floor(number_holes)/2)]

    beamlet_positions = m_per_pixel * (centres - middle)
        # Beamlet locations
    beamlet_widths = widths * m_per_pixel

    if adjust_for_aperture_size:
        # Account for non-negligible aperture size.
        # What width gaussian do I need to convolve with the aperture to get the
        # current demagnified gaussian width?

        # Magnification
        with errstate(invalid='ignore'):
            # Suppress the divide by zero warning.
            magnification = nanmean(beamlet_positions / slit_positions)

        demagnified_beamlet_positions = beamlet_positions / magnification
        demagnified_beamlet_widths = beamlet_widths / magnification

        aperture_f = tophat_f(aperture_size, 0)
        x_range = demagnified_beamlet_widths.max()*10
        xs = linspace(-x_range/2, x_range/2, 1000)
        aperture_y = aperture_f(xs)

        corrected_widths = zeros(demagnified_beamlet_widths.size)
        for i, width in enumerate(demagnified_beamlet_widths):
            result_f = gaussian_f(width)
            result_y = result_f(xs)

            fit_func = lambda x, std: normalise(fftconvolve(gaussian_f(std)(x), aperture_y, mode='same'))

            fit = fitCurve(xs, result_y, fit_func, 4*width)

            fit_ys = fit_func(xs, *fit)

            if False:
                plt.figure()
                plt.plot(xs, result_y, label='result')
                plt.plot(xs, fit_ys, label='fit')
                plt.plot(xs, aperture_y, label='aperture')

                plt.legend()

                print()
                print('Demagnified width:', width)
                print('Fitted width:', fit[0])
```

```
                print()

            corrected_widths[i] = absolute(fit[0]) * magnification

        beamlet_widths = corrected_widths

    mean_divergence = (beamlet_positions - slit_positions) / propagation_distance
    mean_mean_divergence = average(mean_divergence, weights=num_particles)

    rms_divergence = beamlet_widths / propagation_distance

    a = (num_particles*(slit_positions - mean_slit_position)**2).sum()
    b = (num_particles*rms_divergence**2 + num_particles*(mean_divergence - mean_mean_divergence)**2).sum()
    c = ((num_particles*slit_positions*mean_divergence).sum() -\
            num_particles.sum()*mean_slit_position*mean_mean_divergence)

    em = sqrt((a*b-c**2) / num_particles.sum()**2)

    return em

def calc_beam_size(heights, centres, widths, sums=None, pepperpot_pitch=200e-6, \
                   m_per_pixel=m_per_pixel, return_centre=False, diag=False):
    if False:
        print('HAAAAAAAAAX')
        heights = heights[1:]
        centres = centres[1:]
        widths = widths[1:]
        #sums = sums[1:]

    centres_m = centres * m_per_pixel
    widths_m = widths * m_per_pixel
    heights_e = heights / mcp_calibration

    num_particles = heights * widths * sqrt(2*pi) if sums is None else sums
    num_particles /= mcp_calibration
    positions = arange(num_particles.size, dtype='f')
    positions -= positions[int(floor(positions.size/2))]
    positions *= pepperpot_pitch

    middle = average(positions, weights=num_particles)
    std = sqrt((num_particles*(positions-middle)**2).sum()/num_particles.sum())

    fit_func = lambda x, height, centre, width, c: gaussian(x, height, centre, width, c)
    guess = (num_particles.max(), middle, std, 0)

    #middle = positions[heights.argmax()]
    #fit_func = lambda x, height, width, c: gaussian(x, height, middle, width, c)
    #guess = (num_particles.max(), std, 0)


    # What if i add y=0 to x=+/- inf
    ex = zeros(positions.size+2)
    wy = zeros(positions.size+2)
    ex[1:-1] = positions
    ex[0] = positions.min()*100
    ex[-1] = positions.max()*100
    wy[1:-1] = num_particles

    fit = fitCurve(ex, wy, fit_func, guess)
    #fit = fit[0], middle, fit[1], fit[2]

    rms_size_m = fit[2]

    total_beam_count = fit[0]*(rms_size_m**2)*2*pi/(m_per_pixel**2)

    centre = fit[1]

    if diag:
        xs = linspace(positions[0], positions[-1], 1000)
        ys = gaussian(xs, *fit)

        plt.plot(positions, num_particles, '-x')
        plt.plot(xs, ys)
```

```
        plt.axvline(centre, color='r')
        plt.xlabel('x (um)')


    if return_centre:
        return rms_size_m, total_beam_count, centre
    else:
        return rms_size_m, total_beam_count

def refine_gaussian_parameters(xs, line, heights, centres, widths, diag=False):
    params_array = sum(zip(heights, centres, widths), ())
        # Creates a flattened array of parameters, [h, c, w, h, c, w, h, c, w....]

    guess = params_array
    fit = fitCurve(xs, line, n_gaussians, guess)

    refined_heights = fit[0::3]
    refined_centres = fit[1::3]
    refined_widths = fit[2::3]

    partially_successful = False
    for i in range(heights.size):
        # If the new width is much bigger or smaller than the orginal then it's probably wrong.
        if refined_widths[i]>widths[i]*2 or refined_widths[i]<widths[i]/2:
            refined_widths[i] = widths[i]
            partially_successful = True

        # If the new height is much taller or shorter than the original thatn it's probably wrong.
        if refined_heights[i]>heights[i]*2 or refined_heights[i]<heights[i]/2:
            refined_heights[i] = heights[i]
            partially_successful = True

        # If the new centre is many orginal width away from the original centre then it's probably wrong.
        if abs(refined_centres[i]-centres[i])>widths[i]*3:
            refined_centres[i] = centres[i]
            partially_successful = True

    if diag:
        plt.plot(xs, line)

        #plt.plot(xs, n_gaussians(xs, *params_array), '--k')

        smooth_xs = linspace(xs.min(), xs.max(), 5000)

        fit_ys = n_gaussians(smooth_xs, *fit)
        plt.plot(smooth_xs, fit_ys, 'r')

        individual_ys = []
        for h, c, w in zip(refined_heights, refined_centres, refined_widths):
            ys = gaussian(smooth_xs, height=h, centre=c, width=w, offset=0)

            individual_ys.append(ys)

        if partially_successful:
            fit_ys = n_gaussians(smooth_xs, *guess)
            plt.plot(smooth_xs, fit_ys, ':k')
        else:
            ys = refined_heights * refined_widths * sqrt(2*pi)
            ys *= line.max()/ys.max()
            plt.plot(refined_centres, ys, 'rx')

        if False:
            # Paper figure. Overlapping beamlets. Used with Analyse 8.py at column 134.
            # Prepare matplotlib
            from matplotlib import rcParams

            rcParams.update({'font.size': 10})
            rcParams.update({'pgf.rcfonts': False})
            rcParams.update({'pgf.texsystem': 'pdflatex'})
            rcParams['font.family'] = 'serif'
            rcParams['axes.unicode_minus'] = False

            linewidth = 5.71 # inches
```

```
            figwidth = 0.95*linewidth
            figheight = figwidth/3*1.44
            figsize = (figwidth, figheight)

            plt.figure(figsize=figsize)

            xs *= m_per_pixel
            smooth_xs *= m_per_pixel

            xs -= 0.0168
            smooth_xs -= 0.0168

            plt.plot(xs*1e3, line/mcp_calibration, 'b')

            for ys in individual_ys:
                plt.plot(smooth_xs*1e3, (ys - line.max()*2/3)/mcp_calibration, 'r')

            plt.plot(smooth_xs*1e3, fit_ys/mcp_calibration, 'r:')

            plt.yticks([0, 0.05, 0.1, 0.15])

            plt.xlim((-0.01e3, 0.01e3))
            plt.xlabel('Detector Position (mm)')
            plt.ylabel('Electron Count')

            plt.tight_layout()

            plt.savefig('overlapping_gaussians.pgf')

            plt.show()
            exit()

    if partially_successful:
        print('\tRefined parameter search partially unsucessful returning orginals.')
        return heights, centres, widths
    else:
        return refined_heights, refined_centres, refined_widths

# Theory
def expected_emittance(excess_energy, beam_radius):
    T = 2*excess_energy*eV / (2 * kB)
    return beam_radius * sqrt(kB*T/(m_e*speed_of_light**2))

def temperature_from_expected_emittance(emittance, beam_radius):
    T = (m_e*speed_of_light**2) * (emittance/beam_radius)**2 / kB
    return T

def excess_energy_from_wavelength(blue_wavelength, field_ionisation=False, \
                                  electric_field=electric_field):
    red_energy = Plank*speed_of_light/red_wavelength

    blue_energy = Plank*speed_of_light/blue_wavelength

    field_energy = Plank*speed_of_light*4*Ry*sqrt(electric_field/electric_field_atomic_unit)

    if field_ionisation:
        excess_energy = (red_energy + blue_energy) - Rb_ionisation_energy + field_energy
    else:
        excess_energy = (red_energy + blue_energy) - Rb_ionisation_energy

    return excess_energy
```

B.4.2 Fitting.py

```
from scipy.optimize import curve_fit, leastsq
from numpy import ravel, meshgrid, arange, array

def linearSearch(target, func, lower_bound, upper_bound, accuracy, iteration_limit=100):
    def _good_enough(val):
        return abs(target-val)<accuracy

    up_x = upper_bound
```

```
    lo_x = lower_bound

    lo_y = func(lo_x)
    up_y = func(up_x)

    print(up_x, up_y)
    print(lo_x, lo_y)

    print(target)

    if (lo_y<target and up_y>target) or (lo_y<target and up_y>target):
        if _good_enough(lo_y):
            return lo_x

        if _good_enough(up_y):
            return up_x

        i=0
        while(i<iteration_limit):
            # Interpolate
            grad = (up_y - lo_y) / (up_x - lo_x)

            new_x = lo_x + (target - lo_y) / grad
            new_y = func(new_x)

            print(i)
            print(lo_x, new_x, up_x)
            print(lo_y, new_y, up_y)
            print()

            if _good_enough(new_y):
                return new_x

            if grad>0:
                if new_y>target:
                    up_x = new_x
                    up_y = new_y
                else:
                    lo_x = new_x
                    lo_y = new_y
            else:
                if new_y>target:
                    lo_x = new_x
                    lo_y = new_y
                else:
                    up_x = new_x
                    up_y = new_y
            i += 1


    return None


def fitCurve(x, y, fitfunc, p_initial, y_err=None, errors=False):
    """
    Returns the parameters of a 2D distribution found by least squares given
    the fitting function (fitfunc), data (x,y) and initial parameters (p_initial).

    If error=True then the diagonals of the covariant matrix from the fit will
    be returned.

    I believe that the diagonals do not exactly represent the errors on the
    fit but they are related to it. More examination of the least squares
    procedure is required to figure this out.
    """

    p1, cov = curve_fit(fitfunc, x, y, p0=p_initial, sigma=y_err)

    if errors:
        return p1, [cov[i][i] for i in range(len(cov))]
    else:
        return p1
```

```
def fitPlane(x, y, z, fitfunc, p_initial, errors=False):
    """
    Returns the parrameters of a 3D distribution found by least squares given
    the fitting function (fitfunc), data ( z(x, y) ) and initial parameters (p_initial).

    If error=True then the diagonals of the covariant matrix from the fit will
    be returned.

    I believe that the diagonals do not exactly represent the errors on the
    fit but they are related to it. More examination of the least squares
    procedure is required to figure this out.
    """

    errorfunction = lambda p: ravel( (lambda x, y: fitfunc(x, y, *p)) (*meshgrid(x, y) ) -
                                     z)

    p, cov, infodict, mesg, ier = leastsq(errorfunction, p_initial, full_output=True)

    if errors:
        err = [cov[i][i] for i in range(len(cov))]

        return p, err
    else:
        return p

################################################################################

def example_usage():
    # Normal 1D function
    function = lambda x: 3*x + 2

    x = arange(0, 10, 0.1)
    y = function(x)

    fit_func = lambda x, p1, p2: p1*x + p2
    p_guess = (1, 1)

    p = fitCurve(x, y, fit_func, p_guess)

    print()
    print("1D Function:")
    print("Actual p1 and p2:", 3, 2)
    print("Fitted p1 and p2:", p[0], p[1])

    # 2D function
    function = lambda x, y: 3*x + 2*y + 6

    x = arange(0, 10, 0.1)
    y = arange(0, 10, 0.1)
    z = []
    for yval in y:
        row = []
        for xval in x:
            row.append(function(xval, yval))

        z.append(row)
    z = array(z)

    fit_func = lambda x, y, p1, p2, p3: p1*x + p2*y + p3
    p_guess = (1, 1, 1)

    p = fitPlane(x, y, z, fit_func, p_guess)

    print()
    print("2D Function:")
    print("Actual p1, p2, p3:", 3, 2, 6)
    print("Fitted p1, p2, p3:", p[0], p[1], p[2])

    # Heavieside Step Function
    heavieside = lambda x, low, high, threshold: array([(low if ex<threshold else high) for ex in x])

    x = arange(-10, 10, 0.1)
    y = heavieside(x, -1, 3, 2)
```

```
    p = fitCurve(x, y, heavieside, (0, 0, 0))

    print()
    print("Heavieside Function")
    print("Actual low, high, threshold:", -1, 3, 2)
    print("Fitted low, high, threshold:", p[0], p[1], p[2])


if __name__ == "__main__":
    example_usage()
```